Interfacing neural network chips with a personal computer

master thesis of J.J.M. van Teeffelen

supervisor: prof.dr.ir. W.M.G. van Bokhoven
coach: dr.ir. J.A. Hegt
period: January - August 1993

Eindhoven University of Technology
Faculty of Electrical Engineering,
Electronic Circuit Design Group
August 1993

Abstract

The research in the field of neural networks is no longer restricted to theoretical analysis or simulation of these networks on serial computers. More and more networks are implemented on chips, which is of crucial importance if full advantage of the neural networks is wished to be taken when using them in real time applications like speech processing or character recognition.

The Electronic Circuit Design Group at the Eindhoven University of Technology currently is implementing several neural networks with a multi-layered perceptron architecture together with their learning algorithms on VLSI chips. In order to test these chips and to use them in an application they will be connected with a personal computer with help of an interface.

This interface, that has to be as versatile as possible, meaning that it must be able to connect all kinds of neural network chips to it, can be realized either by making use of commercially available interfaces or by designing an own interface with help of off-the-shelf components. Two interfaces will be discussed, one for the rather slow AT-bus and one for the high speed VESA local bus.

Although the commercially available interfaces are not as versatile as wished, and the prices may seem rather high, they turn out to be the best way to realize the interface at the moment. They are guaranteed to work and can be used immediately. The discussed interfaces for the AT-bus and the VESA local bus still have to be tested and implemented on a printed circuit board.

Contents

List of figures

List of figures

1 Introduction

The functioning of the brain has occupied mankind for centuries. There has been a lot of research to gain more insight in the processes that are taking place in our brain. The densely interconnected nerve cells present in our brain can perform difficult tasks like speech recognition and processing visual information much better than the most advanced computers. Artificial neural networks, simplified models of these nerve cells, are a better alternative than traditional computers with their sequential execution of instructions when tackling problems of which the exact solution is not known or the mathematical description of the solution is very complicated and difficult to implement on a computer.

The brain has several features that are desired to be present in artificial neural networks. It is robust and fault tolerant. The death of nerve cells does not decrease the performance significantly. It is flexible, capable of adapting to new situations by learning, in contrast to a computer that has to be reprogrammed in such a case. It can deal with fuzzy, probabilistic, noisy or inconsistent information. It works in a highly parallel manner and it is small, compact and dissipates very little power.

The history of neural networks started in 1943 when a simple model of a neuron as a binary threshold unit was proposed by McCulloch and Pitts. These threshold networks were the main subject of research for the next 15 years. Around 1960 the research concentrated on networks called perceptrons that were investigated by the group of Rosenblatt. In these networks, the neurons were organized in a layer with feed forward connections from the inputs to that layer.

The fact that some elementary computations could not be done with a one-layer perceptron, and there was no learning algorithm to determine the weights in a multi-layered perceptron so that it could perform a given computation simmered the research of these networks for about 20 years. Still people kept working on the development of learning algorithms and the invention of the back propagation algorithm, first by Werbos in 1974 and then independently rediscovered by Parker in 1985 and Rumelhart, Hinton and Williams in 1986, revived the interest for the perceptron networks.

Almost everything in the field of neural computation has been done by simulating the networks on serial computers, or by theoretical analysis. The implementation of neural networks on VLSI chips has been staying behind for years, mainly because of technology reasons. Current research however is also focused on the implementation of several networks on chips. Efficient hardware is crucially important if the full advantage of the neural networks is wished to be taken when using them in real time applications like speech processing or character recognition.

The Electronic Circuit Design Group at the Eindhoven University of Technology is implementing several neural networks with a multi-layered perceptron architecture together with their learning algorithms on VLSI chips. To test the realized chips and to use them in an application they will be connected to a personal computer. In this thesis the design of an interface that is needed to accomplish this will be treated. This interface has to be as versatile as possible. It must be able to interface several different chips with a personal computer without having many changes to be made to the interface.

The design of such an interface will be treated later on in this thesis. First a short introduction into the perceptron networks together with their training algorithms will be given. Then the specifications of the interface will be formulated by investigating some existing hardware implementations of neural networks. On the basis of these specifications and a description of the personal computer the design of the interface will be treated.

2 Introduction to neural networks

2.1 Basic model of a neuron

The brain is composed of about 10^{11} neurons of different types. These neurons are interconnected with tree-like networks of nerve fiber. Signals are transported from one neuron to another through the axon, a single long fiber, which eventually branches into strands that are connected to the synapses of other neurons. If the signals that are received by the synapses reach a certain level, the neuron is activated and transmits a signal along its axon. In figure 2.1 a model of a neuron is shown as it is used in the artificial networks.

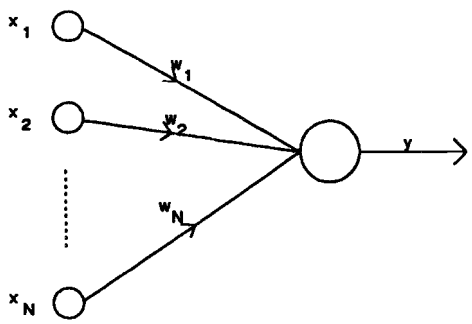


Fig. 2.1: Basic model of a neuron

The neuron computes the weighted sum of the inputs x_i, which can be binary or continuous-valued, and outputs a signal y according to a certain transfer function f:

$$y = f\left(\sum_{i=1}^{i=N} w_i x_i - \theta\right) \tag{2.1}$$

with θ a certain bias. This bias can also be modeled as an input x_0 with value -1 and connected to the neuron with a connection strength w_0 equal to θ. The output of the neuron than equals:

$$y = f\left(\sum_{i=0}^{i=N} w_i x_i\right) \tag{2.2}$$

An often used transfer function is the sigmoid function which is defined as:

$$f_\beta(h) = \frac{1}{1+e^{-2\beta h}} \tag{2.3}$$

with β the steepness parameter. In figure 2.2 an example is given of this sigmoid function with three different values for the parameter β ($\beta_1 > \beta_2 > \beta_3$).

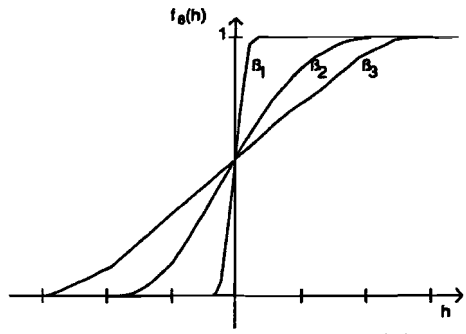


Fig. 2.2: Sigmoid function $f_\beta(h)$

There are two ways to learn the network (change the weights w) to perform a certain task:

* Supervised learning. In this case the learning is done on the basis of a comparison of the output of the network with known correct answers.
* Unsupervised learning. In this case the network is expected to form output classes without additional information about the correct classes.

After the training phase is completed, the network will be able to generalize to new situations. It then can produce correct outputs for inputs it has never seen before. At least, this is the purpose of the training phase. The topology of the network and the number of training iterations that are needed to learn a network will be related to the application it is used in. Next a particular architecture, the multi-layered feed forward networks, also known as multi-layered perceptrons, together with some training algorithms will be described.

2.2 Multi-layered perceptrons

In layered feed-forward networks, also known as multi-layered perceptrons, the network is divided into several layers that are connected in a feed-forward manner. The outputs of neurons in one layer are only connected to inputs of neurons in the next layer. Figure 2.3 shows an example, a two-layer perceptron. In this figure also the notational conventions are shown. The inputs of the neural network are denoted by x_i. Outputs of neurons in the hidden layer (hidden layers are the layers between the inputs of a neural network and the output layer) are denoted by v_j. The outputs of the neurons in the second layer which are the outputs of the network are referred to as y_k. Weights connecting layer i to layer j (i<j) will be referred to as w_{ji}. Note that the inputs of the network are not considered as a layer. The bias factors θ are modeled as extra inputs with value -1 as mentioned before.

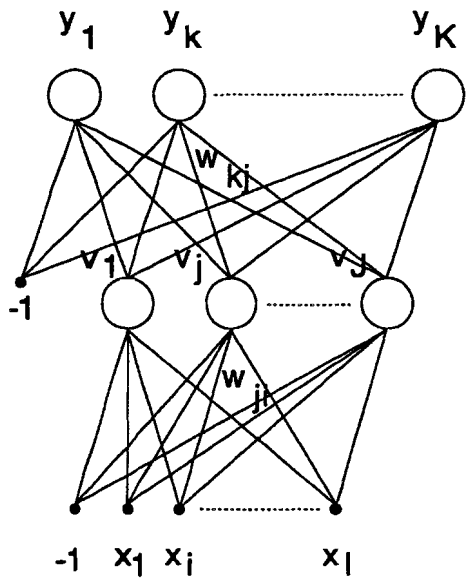


Fig. 2.3: A two-layer perceptron

The weights can be updated in several ways. The Electronic Circuit Design group currently is examining the update of weights according to the back-propagation and weight perturbation algorithms. These two methods will be briefly described in the following. More detailed information about update algorithms in general and the mentioned algorithms can be found in [2], [11], [14], [16], and [18].

2.3 Back-propagation

One method to determine new weights is to use a gradient descent learning algorithm. In this case an error measure or cost function E[w] is defined by:

$$E[w] = \frac{1}{2}\sum_{\mu k}(d_k^\mu - y_k^\mu)^2 \tag{2.4}$$

with μ indicating one of the M input patterns, d_k^μ the desired output of neuron k and y_k^μ the actual output of that neuron. Given this error function, the set of weights w can be improved by sliding downhill the surface that E[w] defines in w space. Specifically, weight w_{kj} is changed once every M patterns by an amount Δw_{kj} proportional to the gradient of E at the present location:

$$\Delta w_{kj} = -\eta\frac{\partial E}{\partial w_{kj}} \tag{2.5}$$

with η representing a certain learning rate. In the case of the two-layer perceptron as shown in figure 2.3 this yields the following results. The error E[w] becomes:

$$E[w] = \frac{1}{2}\sum_{\mu k}\left[d_k^\mu - f\left(\sum_j w_{kj}v_j^\mu\right)\right]^2 = \frac{1}{2}\sum_{\mu k}\left[d_k^\mu - f\left(\sum_j w_{kj}f(h_j^\mu)\right)\right]^2 \tag{2.6}$$

with h_j^μ the total input to neuron j in the hidden layer:

$$h_j^\mu = \sum_i w_{ji}x_i^\mu \tag{2.7}$$

The change for the weights between the hidden layer and the output layer is given by:

$$\Delta w_{kj} = \eta\sum_{\mu k}[d_k^\mu - f(g_k^\mu)]\,f'(g_k^\mu)\,f(h_j^\mu) \tag{2.8a}$$

$$= \eta\sum_{\mu k}\delta_k^\mu f(h_j^\mu)\ ,\quad \delta_k^\mu = f'(g_k^\mu)\left(d_k^\mu - f(g_k^\mu)\right) \tag{2.8b}$$

with g_k^μ the total input to neuron k in the output layer:

$$g_k^\mu = \sum_j w_{kj}\,f(h_j^\mu) \tag{2.9}$$

The weights between the inputs of the network and the hidden layer are changed according to:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \sum_{\mu k} \frac{\partial E}{\partial v_k^\mu} \frac{\partial v_k^\mu}{\partial w_{ji}} \tag{2.10a}$$

$$= \eta \sum_{\mu k} [d_k^\mu - f(g_k^\mu)] f'(g_k^\mu) w_{kj} f'(h_j^\mu) x_i^\mu \tag{2.10b}$$

$$= \eta \sum_{\mu k} \delta_k^\mu w_{kj} f'(h_j^\mu) x_i^\mu \tag{2.10c}$$

$$= \eta \sum_{\mu} \delta_j^\mu x_i^\mu \ , \quad \delta_j^\mu = f'(h_j^\mu) \sum_k w_{kj} \delta_k^\mu \tag{2.10d}$$

As can be seen in (2.10d) the error of the output layer is propagated back through the network. This back-propagation of errors can be easily extended for networks with more than two layers following the same procedure as in (2.6), (2.8) and (2.10). The back-propagation algorithm now does the following:

1. initialize the weights with random values;
2. present input vector x_i^μ and desired output vector d_k^μ to the network;
3. determine the output y_k^μ, and the error δ_k^μ;
4. determine the deltas for the hidden layers by propagating the error backward according to (2.10d);
5. go back to step 2 and repeat for the next pattern until all M patterns are presented;
6. update the weights of the network by an amount Δw according to (2.8) and (2.10);
7. repeat by going to step 2 until the error has reached a desired value.

Although the algorithm is described with an update rate of once per M patters, the update usually is done after each input pattern. The calculation of the derivative of the transfer function f, turns out to be very simple in case of the sigmoid function (3). The derivative then namely equals:

$$f'(h) = 2\beta f(1-f) \tag{2.11}$$

2.4 Weight perturbation

Another method to update the weights of a network is weight perturbation. This also is a gradient descent method, only here the gradient is not calculated but approximated. By disturbing a weight w_{ji} with a small perturbation $pert_{ji}$ and using the forward difference method the weight update Δw_{ji} is given by:

$$\Delta w_{ji} = -\eta \frac{E(w_{ji}+pert_{ji}) - E(w_{ji})}{pert_{ji}} \tag{2.11}$$

The error E usually is the mean square error according to (4). When a better approximation is desired, the central difference method can be used resulting in an update Δw_{ji} equal to:

$$\Delta w_{ji} = -\eta \frac{E\left(w_{ji} + \frac{pert_{ji}}{2}\right) - E\left(w_{ji} - \frac{pert_{ji}}{2}\right)}{pert_{ji}} \tag{2.12}$$

The update of the weights is done in the following way (when using the forward difference method):

1. initialize the weights with small random values;
2. present input pattern and determine the output error $E[w_{ji}]$;
3. disturb weight w_{ji} by an amount $pert_{ji}$;
4. present the same input pattern again and determine $E[w_{ji}+pert_{ji}]$;
5. update weight w_{ji} according to (10);
6. repeat by going to step 2 until the error has reached a desired value.

As in the case of the back propagation algorithm the error E can also be determined after M input patterns, instead of after each pattern as is done in the given procedure.

3 Specifications for a neural network interface

The implementation of neural networks in hardware has been staying behind for years, mainly because of technological constraints. Yet, if these networks are wanted to be used in real applications like processing visual information, it is a prerequisite to implement them in hardware. Optimum benefit can only be acquired when data actually is processed in a highly parallel way, and this again can only be done efficiently in hardware.

Although this field of research still is in a beginning phase, more and more chips exhibiting desired features in a neural network are becoming available. To be able to state requirements for an interface, some chips that were connected to a computer in some way (not necessarily a personal computer), have been examined in literature. The features of these chips form, together with a short description of the chips that are being developed by the Electronic Circuit Design Group the basis for the specifications of the interface.

3.1 Existing hardware implementations

The following aspects are of importance when looking at hardware implementations:
1. architecture of the network;
2. kind of implementation;
3. processing speed;
4. training of the network.
These aspects will be clarified in the following.

3.1.1 Architecture of the network

The topology of multi-layered perceptron chips can be:
1. fixed. In this case a fixed network architecture, e.g. a single layer perceptron, is implemented on a chip. Extension of the network may be possible by interconnecting several chips. Examples of these networks can be found in [7], [12], [13], and [22].

2. reconfigurable. In this case a number of basic neurons with a certain number of inputs and synapses is implemented on the chip. The topology of the network on this chip can

be altered by the user e.g. by changing the contents of some registers ([9], [21], [24], [25], and [33]). Extension of the network to a larger one may also be possible by interconnecting several chips.

The number of neurons and weights that are present on the chip differs in each implementation. In [21] only one neuron is present on the chip, while in [33] 288 neurons can be found. The number of synaptic weights in the examined chips differs from 1024 ([20]) to 262144 ([9]).

3.1.2 Kind of implementation

The kind of implementation can be:

1. digital. All signals are digital in this case (see [6], [20], [21] and [33]). Data enters and leaves the chip through a digital bus, is processed by digital components on the chip and the chip is controlled with digital control lines.

2. analog. All signals, besides a few digital control lines, are analog (current or voltage) in this case ([2], [4], [12], [13], [22], [23], [25]). Data is processed in an analog way on the chip by analog components, e.g. analog multipliers. The weights are usually stored in off-chip RAM and special circuitry is needed to refresh the on-chip weights. All chips that are being developed by the Electronic Circuit Design Group fall into this category.

3. mixed digital-analog. In this case data is processed both in a digital and an analog way ([7], [24]). Inputs and outputs of the chip, as well as the control lines, usually are digital. Data enters the chip via shift registers. Only inside the chip operations are done in an analog way, e.g. the multiplication of the inputs with the weights is performed with analog multipliers.

4. optical. Data can also be processed using optical signals. However, because of the completely different nature of these signals, chips using them will be left out of consideration.

The resolution of the weights and the neurons is problem dependant. Variations between 1 bit and 16 bit are encountered in the mentioned articles.

3.1.3 Processing speed

Speed is an important aspect in the neural net chips. Processing of data during normal operation and updating weights in the learning phase should be done as fast as possible. The speed of the digital chips mainly is determined by the clock frequency at which the chips operate (e.g. 15 MHz in [21]). In analog chips the settling times of the various components determine the speed (the chip in [12], [13] and [22] for example has a maximum processing delay of 3μs per layer in normal operating mode).

3.1.4 Training algorithms

The training algorithm can either be:
1. implemented on the chip;
2. run on a host computer.

The first option places great demands on the hardware, but results in faster training of the network (an example can be found in [33]). The second option on the other hand requires number crunching computers. Training with a host computer can be done in the following ways:

1. chip in loop training. After presenting inputs to the chip, new weights are calculated on the host computer and changed on the chip, on the basis of the outputs that are generated by the chip (see e.g. [12], [13], [22], and [25]) . This kind of training is preferable since the neural net chip processes data much faster than a general purpose computer. Only when the weights of the network can be changed difficultly (meaning it takes too much time to change them), the next method will be chosen.

2. simulation on host. In this case the complete network is simulated on the host computer in the training phase (e.g. [9]). When the training is completed, the weights are loaded on the chip that resumes operation in normal mode.

3. a combination of the methods 1. and 2. First the weights of the network are determined by simulating the complete network on the host computer. Then a sort of fine-tuning is performed by executing a few chip in loop training iterations.

3.1.5 The Intel 80170NX Electrically Trainable Neural Network Chip

One chip that is especially interesting, since it is commercially available, is the Intel 80170NX Electrically Trainable Neural Network chip ([12], [13], and [22]). The features of this chip are already roughly mentioned in the foregoing (paragraphs 3.1.1. to 3.1.4). In figure C.2 (Appendix C) a general overview of this chip is shown. Here, also more precise data on some signals can be found.

The chip contains 64 neurons and 10,240 individually addressable synapses with on-chip storage of weights in EEPROM. A maximum of 128 inputs can be led to the 64 neurons in a feedback mode (64 inputs at a time). The gain of the sigmoids can be controlled externally (with the V_{GAIN} signal). The sigmoids can also be used as a comparator for 0 V or 5 V output (TTL-compatible operating mode). High programming voltages are needed to change the weights on the chip. The maximum processing delay of the chip is 3µs.

Since the Electronic Circuit Design Group does not have any neural networks implemented in hardware at its disposal at the moment, an interface that will be used to control future chips must also be able to control the Intel 80170NX so it will be possible to test the interface. This, however does not mean that all features of this chip must be used by the interface.

3.2 Chips under development

The Electronic Circuit Design Group currently is developing two neural net chip-sets. The first one is a chip-set, with the neurons and synapses implemented on different chips. The back-propagation algorithm, explained in paragraph 2.3, is implemented on-chip, meaning that a backward path will be present on the chip that can propagate the errors of the outputlayer back through the chip. The errors will be calculated by the host computer.

The exact specifications of this chip-set are not known at the time being. All that is certain is that the chips are completely analog. The neuron chips will have a certain number of analog (pulsed-current) inputs and analog outputs, and the synapse chips will contain a certain number of analog weights that cannot be addressed individually. A complete neural network can be made by interconnecting several chips. The processing speed probably will be less than 1.5μs per layer. More detailed information can be found in [4] and [23].

The other chip-set is suited for the weight perturbation algorithm, as explained in paragraph 2.4. Again the exact specifications are not known at the time being. This analog chip-set, also with the neurons and synapses on different chips, will accommodate a certain number of analog (voltage) inputs and analog outputs, a topology that can be determined by interconnection of chips, and a processing speed of probably less than 1.5μs per layer.

The weights of this chip-set are stored in off-chip RAM, and special circuitry is needed to refresh the on-chip capacitors that hold these weights. The use of RAM results in individually addressable weights. The output error will be determined by the host computer. The way in which the weights will be perturbed still is not known. This can be done either by the host computer or by dedicated circuitry (see [2] for more information on this chip-set).

3.3 Specifications for neural interface

In the foregoing some features of existing neural net chips and chips that are being developed have been examined. It is clear that an interface that must be able to connect these chips to a personal computer at least must have:

* a number of analog data input and data output channels;
* a number of digital data input and data output channels;
* a number of digital and/or analog control lines.

The number of digital and analog lines should be as high as possible, since a single chip can have as many as 64 inputs and 64 outputs ([12], [13], and [22]). The speed at which data is transported to and from the chip should be as high as possible since the neural net chips process data much faster than a computer.

Many operations involved in controlling neural network chips are specific to these chips. That is why no dedicated circuitry, e.g. to shift data into a chip, can be placed on the interface. Besides the requirements imposed by the neural network chips, the interface should comply with two extra requirements:

* it must be designed with off-the-shelf components;
* it must exhibit a reasonable cost to performance ratio. In practice this means that the components have to be as cheap as possible, and that the area that is occupied by these components should be as small as possible (the costs of a printed circuit board form a very substantial part of the total costs of the interface; it is very well possible that the board costs more than the components that are placed on it).

In first instance the interface now should exhibit the following:
1. 32 analog voltage inputs and 32 analog voltage outputs, with adjustable ranges;
2. 4 analog voltage control lines;
3. 32 digital inputs and 32 digital outputs;
4. 8 digital control lines;
5. 12 bit resolution for analog lines;
6. less than 10 μs processing time for 32 analog channels. The processing time is the time needed to transfer digital data from the host to the interface, perform the D/A conversion of thirty-two channels, perform the A/D conversion of thirty-two channels and transfer the resulting digital data back to the host.

A hardware design of an interface should:
1. occupy as little area as possible;
2. be made with off-the-shelf components;
3. cost not more than fl 2,500.

Above specifications are set up a little bit arbitrarily, on basis of the examined articles and ideas living in the Electronic Circuit Design Group. For the analog lines, voltages are chosen. If needed these can be converted into currents. The update of weights can be done either by the personal computer, or by a dedicated processor on the neural interface, whatever turns out to be the most convenient. Still, an interface that meets these specifications should be able to control several completely different neural net chips, albeit partially (the Intel 80170NX cannot be controlled completely by an interface with these specifications. Special circuitry will be needed to generate the programming voltages to update the weights, and to use all of the sixty-four analog inputs and outputs).

4 The personal computer

The neural network chips eventually must be able to communicate with a personal computer. The personal computer (PC) will be an IBM compatible computer with an 80386 or 80486 microprocessor. Three features of this computer will be described in the following. First of all the memory organization will be amplified on. Next, two busses that can be present in the computer will be described , and last of all something will be said about the software running on the computer.

4.1 Memory organization

4.1.1 Main memory

The original PC with a 8086 microprocessor could address 1,048,576 unique 8 bit memory locations. Because the 8086 had 16 bit registers, the 20 bit physical addresses were generated by multiplying the contents of a segment register by 16 and adding the contents of an offset register to the result (the addresses are referred to as segment:offset, e.g. A000:0010 represents physical address A0010). In this way the address space is divided into 64K blocks of memory. In figure 4.1 an overview is given of the memory of the original PC. The segment addresses are numbered from 0000 to F000.

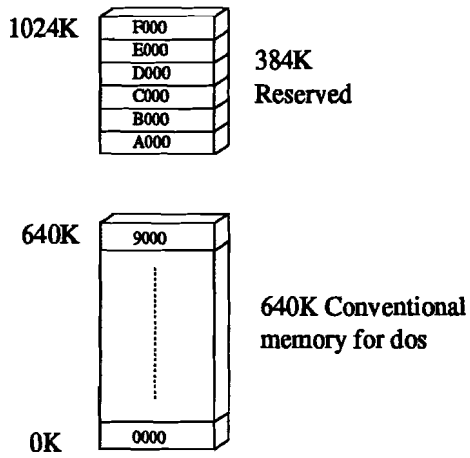


Fig. 4.1: Memory of original PC

The lowest 640K of memory can be used by the operating system (DOS) to run programs in. The memory between 640K and 1024K is reserved for the system. In this area several ROM blocks (C0000-CFFFF is reserved for video ROM, F0000-FFFFF is reserved for ROM

BIOS), and the video RAM (A0000-BFFFF is reserved for this memory) can be found. Segment E (E0000-EFFFF) sometimes is used to set up a page frame. Through this page frame expanded memory, present on a peripheral card, can be addressed, 64K at a time. Physical addresses of memory places not in use in the reserved area actually are wasted.

The 80386 and the 80486 inherited the segmented memory scheme as described before. This memory also still is byte oriented. The reserved area of 384K still is reserved area. Only more memory can be addressed by the 32 bit processors with their 32 bit address busses and more operating modes are available. The memory above 1024K is called the extended memory. The physical limit is 4Gbytes, but it will take a long time before a computer will be equipped with such an amount of memory. The 80386 and 80486 can operate in the following modes:

* real mode. In this mode the processors operate as a 32 bit version of the 8086 using the previous mentioned segmentation scheme. Yet some 32 bit extensions are possible since the operands and addresses are allowed to be 32 bit.

* protected mode. In the protected mode the CPU can address more than 1M of physical memory space and facilities are offered to maintain data integrity in a multitasking environment.

* virtual 8086 mode. This mode can be used to have the processor imitating several real mode 8086 processors running at the same time.

Other changes in 80386 and 80486 with regard to the 8086 are the segmentation and paging schemes allowing programmers to address 4Tbytes of logical addresses. These logical addresses do not correspond directly with the physical addresses anymore as they did in the 8086. More detailed information about these features can be found in [3] and [17]. It must be noted that no matter how much memory is present, DOS can only access the first megabyte of it.

4.1.2 Shadow RAM

Most new computers based on a 80386 or 80486 have a user option to copy the contents of slow ROM into an area of extra onboard RAM. This area is called shadow RAM. When DOS tries to access the ROM blocks, a pointer now refers to the shadow RAM, instead. This shadow RAM usually is mapped somewhere in the reserved memory area.

4.1.3 Cache memory

Besides the main memory newer 80386 computers also have a cache memory, fast memory that holds blocks of data (typically 2, 4, 8 or 16 bytes) of the slower main memory. The 80486 computers usually also have this external cache memory in addition to the on chip cache. This internal cache of the 80486, capable of storing 8K of code and data in 16 byte blocks is a fully associative cache, with write-through memory update. This cache can be disabled and flushed in software. Flushing the internal cache also results in flushing the external cache in a 80486 computer. In a 80386 computer the external cache cannot be flushed by software since the 80386 has no instruction to do that.

4.1.4 I/O

External devices can be addressed with:

- available isolated I/O addresses. The 80386 and 80486 allow for 64K I/O addresses, which can be mapped on 64K 8 bit ports, 32K 16 bit ports or 16K 32 bit ports. Special instructions are available to input and output data of these ports. It must be noted that the I/O addresses 0000-03FF usually are in use by the system, leaving 64,512 addresses to be used by additional I/O devices.

- memory mapped I/O addresses. In this case the external devices respond to ordinary memory addresses. All instructions can be used on these addresses allowing programming flexibility. Care has to be taken when using this method in combination with a cache. If new data is read from an external device, data is read out of the cache if the address is present, instead. This problem can be solved by flushing the cache before reading a memory mapped I/O device or by excluding the memory that is occupied by the I/O device from the cacheable memory.

4.2 The AT-Bus

4.2.1 Introduction

Although the bus that can be found in the current personal computers has been given the name Industrial Standard Architecture bus (ISA-bus) one could hardly speak of a standard until recently. This may be explained by the fact that the ISA-bus is not a true bus in the narrow definition of the word. Unlike other standard busses, this bus is designed around a specific processor family (the Intel 80x86) rather than an universal architecture.

To stop the proliferation of chip-sets and peripheral cards with their own specifications that are all slightly different, the Institute of Electrical and Electronic Engineers decided on recommendation P996 in 1990. And even though the P stands for preliminary this really is a step forward. In the following the specification of the AT-bus according to IEEE P996 will be described. More detailed information can be found in [27] and [28].

4.2.2 AT-bus signals

In figure A1 (Appendix A) the pin identification and the signals of the AT-bus are shown. The AT-bus is a mainly asynchronous bus with some synchronous components. It is meant to deal with memory and I/O accesses to and from peripheral devices. The AT-bus supports the following buscycles:

1. CPU - memory, transfer of data between the CPU and memory;
2. CPU - I/O, transfer of data between the CPU and I/O;
3. Busmaster - memory, transfer of data between a busmaster and memory;
4. Busmaster -I/O, transfer of data between a busmaster and I/O;
5. DMA - I/O and memory, transfer of data between peripheral components and memory or I/O on a basis of Direct Memory Access;
6. Refresh, cycle needed to refresh the dynamic memory chips.

The first five cycles can be further divided into:
1. 8 and 16 bit;
2. read and write;
3. standard, ready and 0 waitstate cycles.

The signals on the bus will be briefly described in the following. Active low signals are preceded by /.

/0WS, Zero Waitstate.
The zero waitstate signal is used to indicate that the buscycle can be completed without the insertion of waitstates. /0WS is the only signal that is synchronous to the bus clock.
AEN, Address Enable.
Address enable allows a DMA controller to take over the busses. During a DMA transfer this signal remains high, prohibiting I/O ports of responding falsely to the memory addresses present on the bus.
BALE, Bus Address Latch Enable.
The falling edge of BALE indicates that the latched addresses SA0..SA19, AEN and /SBHE are valid. During a DMA transfer BALE must be high during the entire buscycle.
/BCKL, Bus Clock
The bus clock may vary between 6 and 8 MHz with a duty cycle of 50% (±5%).
DRQ0,1,2,3,5,6,7, DMA Request Channel x,
/DACK0,1,2,3,5,6,7, DMA Acknowledge Channel x.
A DMA transfer is requested with the DRQx signal. After an acknowledge with /DACKx, the DMA controller can take over the busses, and perform the transfer.
/IOCHK, I/O Channel Check.
Errors that occur on a peripheral card, e.g. a parity error, can be reported to the CPU by taking /IOCHCK low.
IOCHRDY, I/O Channel Ready.
Waitstates can be inserted on the bus by deactivating IOCHRDY. All necessary signals then remain on the bus for a time between 125ns and 15.6µs.
/IOCS16, I/O Chip Select 16 Bit.
This signal is used to indicate that the I/O access will be a 16-bit access.
/IOW, I/O Write,
/IOR, I/O Read,
/MEMW, Memory Write,
/MEMR, Memory Read,
/SMEMW, Small Memory Write,
/SMEMR, Small Memory Read.
The kind of buscycle, a write or read cycle is indicated by these signals. In case of a memory write or read, /SMEMx is only active with addresses in the lowest 1MByte. /MEMx is active for all addresses.

/IRQ3..7, /IRQ9..12, /IRQ14..15, Interrupt Request
Interrupts can be generated with these lines. The interrupts are prioritized, with IRQ9 through IRQ12 and IRQ14 through IRQ15 having the highest priority (IRQ9 is the highest) and IRQ3 through IRQ7 having the lowest priority (IRQ 7 is the lowest).
LA17..LA23, Large Addresses.
These lines form the upper seven address lines of the address bus. They are present on the bus before the small addresses, but unlike these addresses, they are not latched and do not remain on the bus for the entire cycle.
/MASTER, Master.
This signal is used by a busmaster to indicate that it is ready to control the busses.
/MEMCS16, Memory Chip Select 16 Bit.
This signal must be activated by a peripheral card in the case of a 16-bit access. It must be returned in time, requiring fast decoders.
OSC, Oscillator
This is a 14.31818 MHz clock.
/REF, Refresh.
/REF is a signal that indicates a refresh cycle, needed to refresh dynamic memory chips.
RESDRV, Reset Drive.
The reset signal is only active in case of power-up, power supply failure, or system-reset.
SA0..SA19, Small Addresses.
These 20 signals address the lowest 1MByte. They remain on the bus during the entire buscycle.
/SBHE, System Bus High Enable.
This signal is active when data is transferred over the upper eight bits of the data bus (SD8..SD15).
SD0..SD7, System Data Lo-Byte,
SD8..SD15, System Data Hi-Byte.
These signals form the 16-bit wide data bus.
TC, Terminal Count.
Terminal count is used to indicate the end of a DMA transfer. This is done by generating a pulse when the last data transfer is reached.
Power supplies
+5V: 4.875 .. 5.25 V, 3.0/4.5 A, 50mV noise
-5V: -4.5 .. -5.5V, 0.2A, 50mV noise
+12V: 11.4 .. 12.6V, 1.5A, 120mV noise
-12V: -10.8 .. -13.2V, 0.3A, 120mV noise
Gnd: ground

4.2.3 AT-bus timing

The signals that are generated by the buslogic must travel some distance over the mainboard before reaching a peripheral card. Together with the present capacities this results in a delay of about 11ns per signal line when 8 slots are present on the mainboard. So signals returning from the peripheral cards can have additional delays of up to 22ns.

Special attention must be paid to the open collector signals. If an open collector line returns to non active state, it can last a while before this state actually is reached. This time depends on the pull-up resistors and the line capacities. With TTL levels (V_{CC} = 4.5V, V_L = 0.5V, V_H = 2.4 V) the following formula can be used to determine the rise time:

$$Rise\ time = 0.65 * R * C \tag{4.1}$$

Pull-up resistors of 300 Ohm are required for /IOCS16, /0WS, /MEMCS16 and /MASTER. A 1K Ohm pull-up is needed for IOCHRDY. The /IRQx signals use a 2.2K Ohm pull-up and the signals /IOW, /IOR, /MEMW, /MEMR, /IOCHCK and /REF require a 4.7K Ohm pull-up resistor.

In Appendix A the most important timing diagrams (16-bit I/O and 16-bit memory CPU buscycles) are shown. The bus operates at a frequency of 8 MHz although some manufacturers are offering speeds of up to 12 MHz at the moment. At 8 MHz, the maximum data transfer rate that can be attained is 8.00 MByte/s. To complete the description of the bus also the physical dimensions of a peripheral card for the AT-bus are shown in Appendix A. In a hardware design, the lines coming from the bus connector may be connected to not more than two TTL-ports on the peripheral card.

4.3 The Vesa local bus

4.3.1 Introduction

Since the introduction of the personal computer, the performance of this computer kept growing by the introduction of newer, faster microprocessors. The 80486 can deliver 54 MIPS, quite something more than the 8086, which can deliver about 0.75 MIPS. The only component in the PC that kept behind was the bus that formed the connection to the outside world. The only major change was the upgrading of this original 8-bit bus to the previously described 16-bit bus. However, the data transfer rate of this bus (8 MByte/s) in no way satisfies the demands of the current users.

A solution to this problem is the use of a local bus that connects peripherals directly to the CPU. Several manufacturers thought of this and supplied their systems with such a local bus resulting in various different non compatible busses. To stop the development of more of these systems, VESA, the Video Electronics Standards Association, and Intel worked on the development of a standard. Since the Intel bus standard is not available yet, and the Vesa local bus is already being used by many manufacturers, producing mainboards with this bus at small additional costs, only this local bus will be described.

The Vesa local bus (VL-bus) is a full electrical, mechanical, timing and connector specification, allowing high speed peripheral devices to interface, either directly or indirectly, to the local bus of a CPU, providing data transfer rates of up to 130 MByte/s. The bus supports 386 and 486-type CPUs. Other types of CPU can be used but than the signals of that CPU have to be converted to the signals of a 80386 or 80486. In practice however, only 80486-type computers are provided with a Vesa local bus. Figure 4.2 shows the structure of a Vesa local bus system.

In the figure the logical flow of information is shown. A module that resides lower in the hierarchy may not claim ownership of address and data busses if these are claimed by a module with a higher priority.

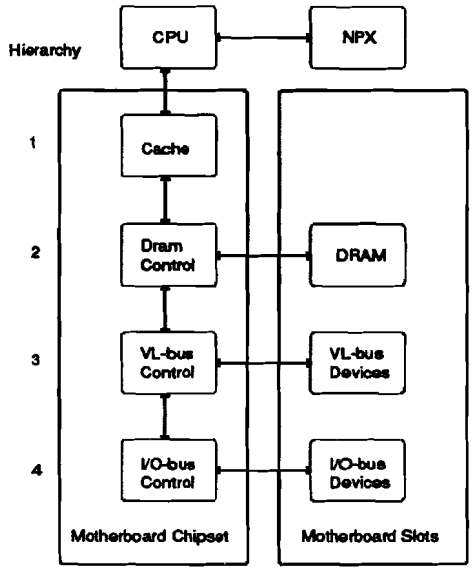


Fig. 4.2: VL-bus architecture

4.3.2 VL-bus signals

The VL-bus is modeled after the 80486 CPU. This means that most of the signals on this (synchronous to the CPU clock) bus are directly related to the CPU signals. In Appendix B these signals are shown together with the pin identification of the VL-bus connector. This connector (a 16-bit micro channel connector) physically resides directly in-line with the ISA connector on the motherboard. In Appendix B, also the physical layout of a VL-bus card is shown.

In the following the signals of the VL-bus will be described briefly. The emphasis will be on 32 bit CPU memory and I/O cycles. Detailed information on other cycles (busmaster, DMA and 16 bit cycles) and more detailed information on the several signals can be found in [31] and [32].

The following abbreviations are used in the description of the signals:
LBC: VL-bus local bus controller. This controller physically resides on the motherboard.
LBT: VL-bus local bus target. This is a device that responds to transfers initiated elsewhere in the system.
Active low signals are indicated with # (and not with / to make a clear distinction between AT-bus and VL-bus signals).

Signals from the system logic.

ID<4..0>, Identifier pins.

A LBT can identify the type and speed of the host CPU with the help of the ID pins, static pins that contain valid data only during power on reset (they should be latched on the trailing edge of RESET#). ID<4> is reserved for future use. The CPU type is identified with ID<1> and ID<0> (a 80386 is indicated with ID<1,0>=01, a 80486 is indicated with ID<1,0>=10, other combinations of ID<1,0> are reserved). ID<2> indicates whether the LBC is capable of handling high speed zero wait state write transfers (ID<2>=1). It can be ignored by the LBT, if it cannot complete a write with zero wait states.The LBT may default to a minimum of one wait states in this case (this mode is indicated with ID<2>=0). Read transfers are not affected by the setting of ID<2>. The speed of the CPU is indicated by ID<3> (ID<3>=1 if speed is less than or equal to 33.3 MHz, ID<3>=0 if the speed is greater than 33 MHz).

LCLK, Local CPU Clock.

The VL-bus clock signal is 1x clock that is in phase with the 486 system clock. The maximum frequency is 66 MHz. CPU state changes are signified with the rising edge of LCLK. The duty cycle of this signal is between 40% and 60%. The high state of LCLK is 2.0V and the low state is 0.8V. The maximum rise and fall times are 2ns. Although the highest specified frequency is 66 MHz, the used VL-bus connector is limited to frequencies of up to 40 MHz. This is why the fastest personal computer with a local bus available at the moment is a computer with a 80486DX2 microprocessor, externally operating at 33 MHz (this is also the frequency at which the bus operates) and internally operating at 66 MHz.

Power, ground, and reserved.

All power and ground pins must be used by a VL-bus device. All power lines V_{CC} are 5V power lines, with a tolerance of 5%. Power must be drawn equally from these power pins. A maximum of 10W may be drawn from a slot by a VL-bus device. Reserved pins may not be used by any VL-bus device.

RESET#, System Reset.

The reset signal is activated after system power up and before any valid CPU cycles take place.

RDYRTN#, Ready Return.

This signal usually is equivalent to the processor RDY# signal. A LBT can recognize the end of a cycle with RDYRTN# .

WBACK#, Write Back.

This signal is reserved for future use with write-back cache systems. LBTs may ignore this signal.

Signals from the CPU

ADR<31..02>, Address Bus.

On this bus the addresses are transferred .

ADS#, Address Data Strobe.

This signal indicates that data on the address bus is valid. ADS# signifies the beginning of every memory or I/O cycle.

BE<3..0>#, Byte Enables.

The data bus is divided into 4 byte lanes. BE<3..0> indicate which lanes are involved in a transfer.

BLAST#, Burst Last.

BLAST# is used to indicate the end of a burst cycle.

DAT<31..00>, Data Bus.

Data is transferred on this 32 bit bus. The valid byte lanes are determined by BE<3..0>#.

D/C#, Data or Code Status.

This signal is used to indicate whether data or code is being transferred on the bus.

M/IO#, Memory or I/O Status.

The type of access, memory or I/O, is indicated by this signal. In case of a memory access M/IO# is high, in case of an I/O access it is low.

W/R#, Write or Read Status.

A write access is indicated by W/R# high, a read access is indicated by W/R# low.

Signals from the VL-bus controller.

LEADS#, Local External Address Data Strobe.

Whenever an address is present on the VL-bus that performs a CPU cache invalidation cycle, this signal is activated. LEADS# is not active for CPU writes.

LGNT<x>#, Local Bus Grant.

A request of a bus master to gain control over the busses (by LREQ<x>#) can be acknowledged with LGNT<x>#. As long as LGNT<x># is asserted the bus master is in control of the busses. Each slot has one pair of LREQ# and LGNT# signals.

LKEN#, Local Cache Enable.

If a VL-bus transfer is cacheable, LKEN# is activated.

Signals from the VL-bus target.

BRDY#, Burst Ready.

BRDY# is used to end the current active burst cycle. This signal also must be synchronized to LCLK. A LBT that doesn't support burst cycles may leave this signal unconnected. If BRDY# and LRDY# are asserted at the same time, BRDY# is ignored and

the remainder of the current burst cycle is concluded as non-burst cycles.

IRQ9, Interrupt Request Line 9.

This interrupt request line, electrically connected to IRQ9 of the ISA bus, is present on the VL-bus for stand alone VL-bus devices, that have no ISA signals available.

LBS16#, Local Bus Size 16.

A LBT that cannot accept 32 bits of data in a single clock cycle can force the CPU to run multiple 16 bit transfers by asserting LBS16#.

LDEV<x>#, Local Device.

A LBT signals the LBC that the current cycle is a VL-bus cycle with LDEV<x>#. Each slot has its own LDEV# signal. All VL-bus devices must drive this signal to valid TTL levels at all times.

LREQ<x>#, Local Request.

LREQ<x># is used to request control of the VL-bus by a device. LBTs that don't act as a bus master must leave this signal unconnected.

LRDY#, Local Ready.

LRDY# is used in the handshake procedure that ends the current active bus cycle. LRDY# is synchronized to LCLK so appropriate setup and hold times to LCLK must be satisfied.

4.3.3 VL-bus timing

In figure 4.3 the general timing of the VL-bus is shown. A CPU transfer starts when valid information is present on ADR<31.02>, M/IO#, W/R#, D/C# and BE<3..0>#. ADS# is strobed to begin the transfer. If a LBT must respond to the address, it has 20ns to assert LDEV#. The assertion of LDEV# prevents the ISA-bus controller to start a cycle.

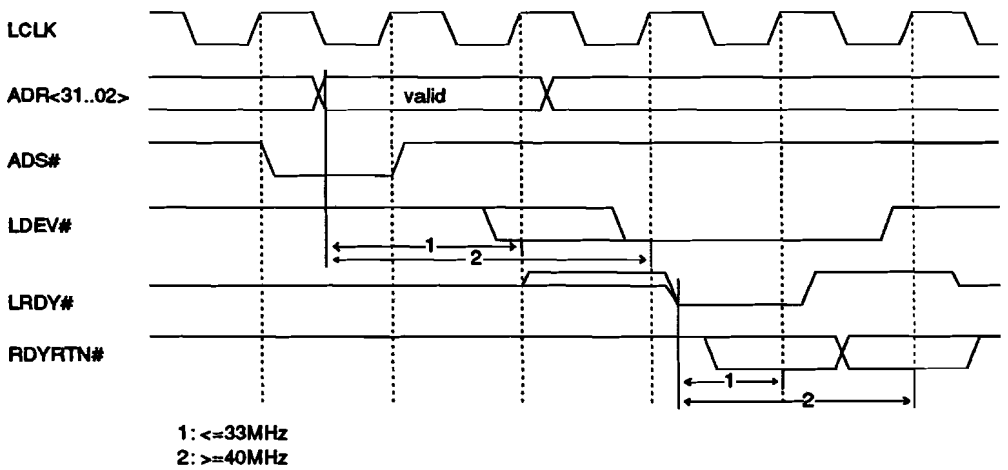


Fig. 4.3: General VL-bus timing

Depending on the speed of the CPU and the VL-bus controller design, LDEV# is sampled at either the LCLK edge following ADS# or two LCLK cycles after ADS#. LRDY# is driven by a LBT after ADS# is high again. After completion of the transfer the LBT asserts LRDY# for one LCLK cycle and then makes it high again for one-half LCLK cycle prior to releasing it. The VL-bus controller responds to the assertion of LRDY# by asserting RDYRTN#. This can be done either immediately or on the next LCLK cycle (in case of speeds greater than 33MHz). If a read transfer is performed, the LBT must hold the read data on the bus until the LCLK on which RDYRTN# is asserted. More detailed timing diagrams involving CPU transfers can be found in appendix B (timing specifications of burst, busmaster or DMA cycles can be found in [31] and [32]).

4.3.4 DC Characteristics

Steady state voltages on the bus may not be higher than V_{CC} and lower than ground. An overshoot over V_{CC} and undershoot under ground may be no more than 0.5V for 5ns. The length of traces from the VL-bus connector to add-in board circuitry is limited to two inches (in case of branched traces, the sum of the branches may be no more than two inches). Each add-in board may have a maximum of one TTL load on each VL-bus input signal. All shared VL-bus signals on an add-in board must be capable of driving a 100pf capacitive load. Non-shared signals, such as LDEV#, must be capable of driving a 20pf load. The signal impedance on each trace, should be equal to or less than 50 Ohm. This signal impedance can be calculated with the following formula:

$$Z_{signal} = \frac{Z_{trace}}{\sqrt{\frac{C_{trace}}{C_{component}} + 1}} \qquad (4.2)$$

with:

Z_{signal} = signal loaded trace impedance;

Z_{trace} = the impedance of the board trace;

C_{trace} = the capacitance of the board trace;

$C_{component}$ = the load capacitance from components and connectors.

The sink current requirements of the output drivers are given in appendix B.

4.4 Software aspects

Programs generally can be written in two ways:
1. using a high level programming language like C;
2. using assembly language.

The first method is the easiest and allows flexible, well-organized programs, while the second method is more difficult, and usually results in less readable programs. On the other hand, the second method provides full control of all present hardware and can result in faster programs. This can be useful when optimum benefit of the hardware resources must be acquired. A middle course can be the use of assembly routines that are incorporated in a program written in a high level language. In this way both flexible programming and fast programs are possible.

Software can be written independently from the bus that is present in the computer. The bus hardware that is present is completely transparent to software. The mode in which the processor operates however influences the way in which physical addresses are generated. In real mode, the logical addresses used in programs, correspond directly to physical addresses (they are equal). In the other modes, the physical addresses usually do not correspond to the used logical addresses, but a translation is performed.

5 Design of an interface

5.1 General survey

5.1.1 General scheme of interface

In figure 5.1 a scheme is given of a complete neural network system, containing a neural network, a personal computer and an interface in between. The task of the interface is to convert the signals of the computer's bus to signals that can be used by the neural network. The personal computer is in full control of the neural network.

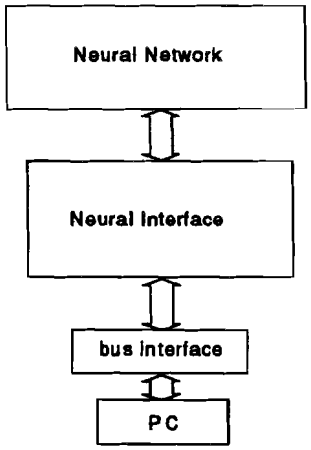


Fig. 5.1: Scheme neural network system

As can be seen in fig. 5.1 the system can be divided into 4 layers:

1. neural network
2. neural interface
3. bus interface
4. personal computer

The neural network is one of the networks as described in chapter 3. The neural interface provides the signals required by the neural network and the bus interface, i.e. digital and analog signals. The third layer, the bus interface, forms the connection to the PC's bus, either an AT-bus or a VL-bus. Finally the computer, a 80386- or 80486-based PC, provides facilities to control the neural network and process data from the network. A more detailed scheme of the neural interface, inspired by test circuits given in [7], [9], [10], [20], [22], and [25], is the scheme shown in figure 5.2.

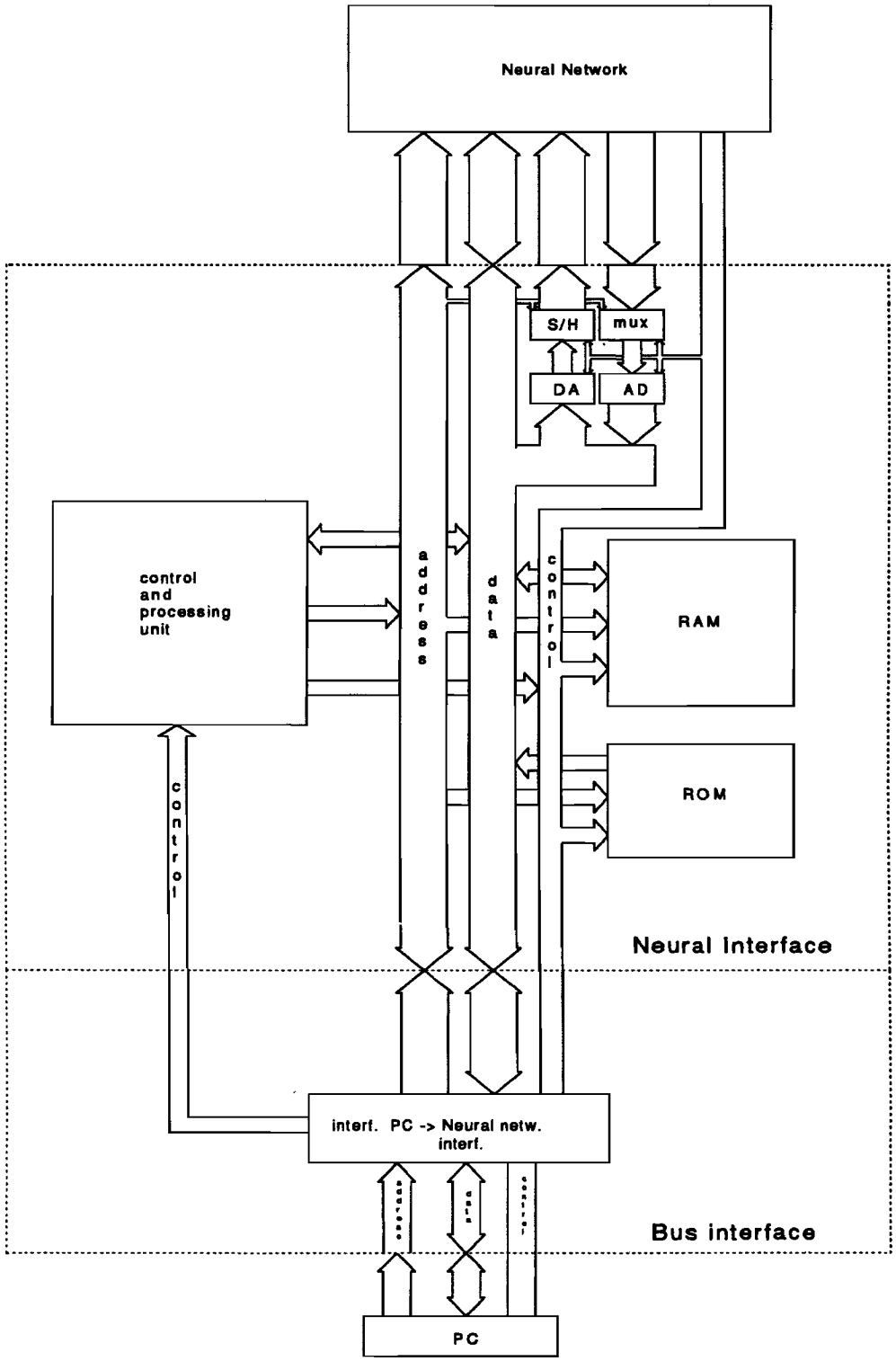


Fig. 5.2: General scheme neural interface

The following components can be found in figure 5.2:

1. control and processing unit. The neural network usually is several times faster than the personal computer. Transferring data between memory and the neural network (the speed at which this is done, is determined by the computer's bus speed) and processing data (updating weights) can be done faster by a dedicated processor, capable of performing floating point calculations, e.g. a digital signal processor. The personal computer then only has to monitor the working of this processor. It downloads programs on the processor and occasionally acquires the results of the simulations on the neural network. A slow bus in the personal computer does not decrease the performance of the neural interface significantly, allowing for high speed operation of the neural network.

2. RAM. This type of memory can be used to store programs for the control and processing unit, data of this unit and data of the neural network (weights, input testvectors, output results and configuration data). Both the control and processing unit on the interface and the personal computer must have access to this memory.

3. ROM. Programs for the control and processing unit, as well as data of the neural network (input testvectors, configuration data) can also be stored in ROM. This can be useful if this kind of data does not have to be changed. The personal computer does not need to have access to this memory.

4. Analog to digital and digital to analog converters together with some multiplexers. These converters are used in case of an analog neural network. The multiplexers can be used to increase the number of analog channels, without adding more converters..

The interconnection of layer 2 and the personal computer is provided by layer 3. This layer converts signals if needed, and provides facilities to address all the components on the neural interface and on the neural network.

A neural interface as shown in figure 5.2 can be realized in two ways:
1. by making use of commercially available interface boards;
2. by designing an own interface board.
These two possibilities will be amplified on in the following.

5.1.2 Commercially available interfaces

Several boards are available as add-in card for a personal computer that exhibit some features of the scheme shown in figure 5.2. A short description of a signal processor board, is given in figure C.1 (appendix C, see also [30]). Efficient processing of data is possible with help of the two parallel data busses and the peak processor performance of 16.7 MIPS and 33.3 MFLOPS. Using such a board has some advantages. It is available at wish, requiring no development time. Also it is guaranteed to function and no basic software routines have to be written. Of course there are also some disadvantages. Not all the desired functions are standard available. The analog channels for example have to be added, requiring development time and additional costs. With regard to the costs it must be noted that these can be rather high. The board shown in fig. C.1 together with the necessary software costs about fl. 10,000. The board of figure C.1 is very useful when trying to achieve a very high speed neural interface, without looking at the costs of it.

Next to the previously mentioned board also more simple boards are available, specialized in acquisition of analog data. These boards only contain some analog channels and are not as expensive as a signal processor board. The number of analog channels however usually is very limited, the conversion speed is not very high (sampling rates of up to 50 KHz are normal for boards costing less than fl. 1,500), and processing of data must be done by the computer. So if the specified number of analog channels (32 inputs and 32 outputs) should be available, probably more than one data acquisition board would be needed, costing much more than the allowed price of fl 2,500. Maintaining the speed requirement results in even more expensive boards, costing more than fl. 4,000 for a thirty-two channel neural interface.

If it is no longer required to use a personal computer, an alternative is offered by the interface described in [15]. This VME based interfaces can be used in conjunction with e.g. a SUN workstation. The processing speed of such a station is higher than that of a personal computer (even when using a 80486). The interface accommodates 64 digital and 64 analog channels. The analog channels can be either configured as input or as output channels. The conversion of 32 analog input channels takes less than 56 µs while the conversion of 32 analog output channels only takes 3 µs.

A disadvantage of the board is the fact that the voltage ranges of the input and output channels are fixed. The output is a voltage between 0 and 5 V, while the input voltage must be between -5 and +5 V. Since the 12 bit resolution is mapped on these ranges, resolution is thrown away if the actual voltage ranges are not equal to the interface's

ranges, or additional circuitry must be used to adapt the voltage ranges of the neural network to those of the interface. The price may be another disadvantage. A complete working system, together with software, costs about fl. 17,000.

Considering the prices of the commercially available boards and the fact that none of these boards meets all the specifications given in chapter three, the design of an own neural network interface board can be a good alternative. The design of an own interface will be described in the following paragraphs.

5.1.3 Design of a board

There are several possibilities to realize an own neural interface board:

1. realization of a board according to figure 5.2, containing a digital signal processor, RAM, ROM (if needed) and the necessary analog channels. Although this method results in a fast, versatile interface, a few remarks must be made. The development of such a board (design, realization and testing) namely takes a lot of time. It is very unlikely that a properly functioning board can be made in half a year. The costs can also grow to an unreasonable height. Although the components themselves need not to be that expensive, the printed circuit board that has to accommodate them, can cost quite a lot of money, since it will very likely be a large multi-layer print.

2. realization of a data acquisition board. In this case only some digital and analog input and output channels are realized. The task that the control and processing unit in fig. 5.2 was meant to perform, now will be done by the personal computer. The speed at which an update algorithm can be executed now completely depends on the speed of the computer. The data transfer speed will be determined by the bus speed and the conversion times of the converters. Advantages of this method are the smaller development time, the smaller size of the printed circuit board and the smaller costs.

Considering the remarks in the foregoing, the second method has been chosen to design an interface at a reasonable price and in a short period of time. The scheme of the interface now changes in the one shown in figure 5.3.

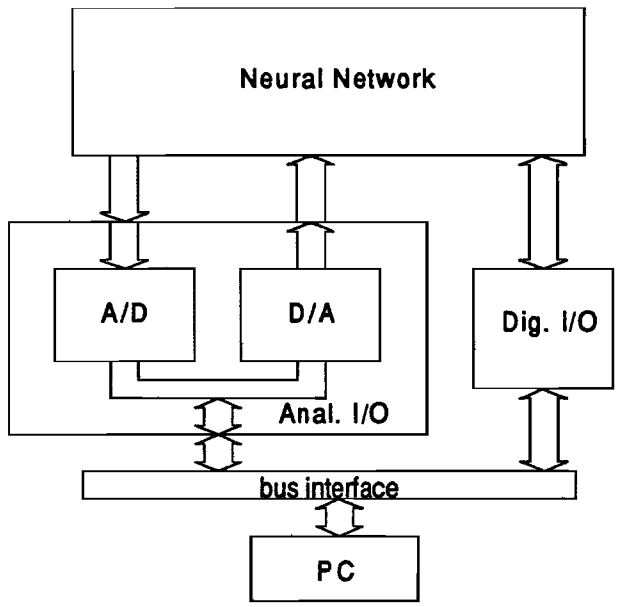


Fig. 5.3: Scheme designed neural interface

In figure 5.3 the second layer of the complete neural network system is divided into two parts:

1. Analog I/O;
2. Digital I/O.

These parts will be covered in more detail in the next paragraphs. The digital I/O will be described together with the bus interface. Actually, two designs of an interface will be discussed. One for the slower AT-bus and one for the high speed VL-bus.

5.2 Analog I/O

The analog I/O block consists of two parts:
1. analog to digital conversion;
2. digital to analog conversion.
These parts will be described separately in the following. At the end of this chapter, a complete analog I/O circuit will be presented.

5.2.1 Analog to digital conversion

The analog to digital conversion will be done with 12-bit Analog to Digital (A/D) converters. There are two basic methods to realize thirty-two analog input channels, shown in the figures 5.4 and 5.5.

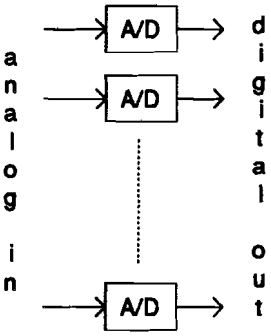


Fig. 5.4: Direct A/D conversion

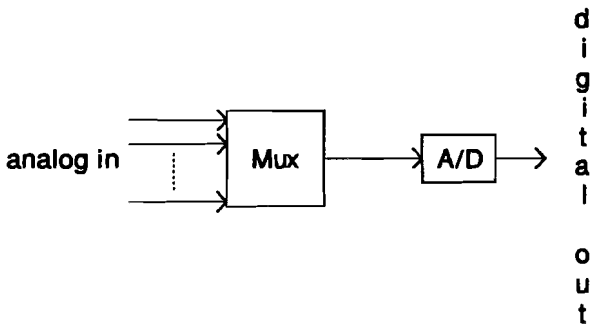


Fig. 5.5: Multiplexed A/D conversion

The first method is rather straightforward. Every analog input channel is realized with one A/D converter (ADC). The second method makes use of multiplexers and requires less ADCs for the same number of analog input channels. Although the first method can result in a faster circuit, it can also be quite expensive. This is caused by the fact that thirty-two ADCs are needed. Not only are the costs of these thirty-two ADCs rather high, but also a large area on a printed circuit board is occupied by these converters, resulting in an even more expensive board.

The second method is less space consuming, but on the other hand it is very expensive to realize a fast circuit in this way (fast ADCs are very expensive, see Appendix C for more information). A middle course, the use of more than one converter together with some multiplexers can form an alternative. This method seems to be the most convenient when trying to realize a circuit with a good price to performance ratio, and therefore this method is chosen (it is cheaper, while still a reasonable conversion speed can be attained).

The circuit will be made up of two ADCs with two 16-channel multiplexers. The converter that will be used is the AD1671JQ from Analog Devices. This 12-bit converter is a true 1.25 MSample/s converter, meaning that it can complete a conversion every 800ns. So theoretically, sixteen channels can be converted in 12.8 µs. The multiplexer is the 16-channel, ADG526AKN from Analog Devices. Since the input range of the A/D converter is fixed, the converter is preceded by an operational amplifier (opamp) circuit to adapt the output voltage of the neural network to the fixed range of the converter. The opamp also acts as an input buffer for the converter. A scheme of sixteen analog input channels is shown in figure 5.6.

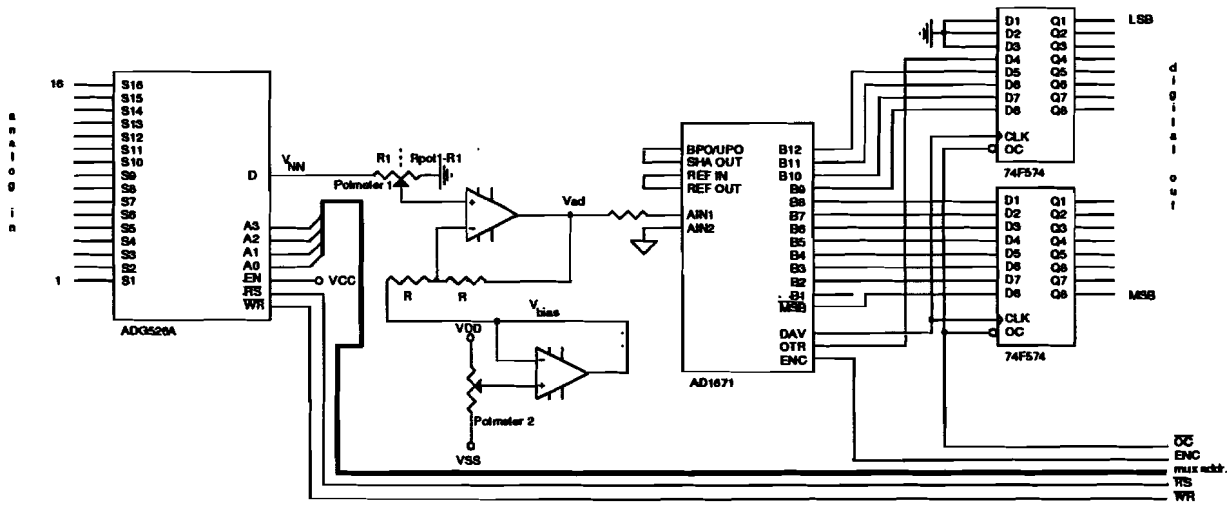


Fig. 5.6: 16-channel analog input circuit

The output voltage of the neural network is denoted by V_{NN}, the input voltage of the A/D converter by V_{AD}. The potentiometers 1 and 2 can be used to adapt the output range of the neural network ($V_L \leq V_{NN} \leq V_H$) to the fixed voltage range of the converter ($0 \leq V_{AD} \leq 5$). The input voltage of the A/D converter is given by:

$$V_{AD} = 2\frac{R_{pot1} - R_1}{R_{pot1}} V_{NN} - V_{bias} \tag{5.1}$$

with R_{pot1} the total resistance of potentiometer 1 and R_1 a part of this resistance. The settings of potentiometer 1 and V_{bias} can be determined with:

$$0 = 2\frac{R_{pot1}-R_1}{R_{pot1}} V_L - V_{bias} \tag{5.2}$$

and

$$5 = 2\frac{R_{pot1}-R_1}{R_{pot1}} V_H - V_{bias} \tag{5.3}$$

yielding:

$$V_{bias} = \frac{5V_L}{V_H - V_L} \tag{5.4}$$

and:

$$\frac{R_1}{R_{pot1}} = 1 - \frac{2.5}{V_H - V_L} \tag{5.5}$$

The output range of the neural network must satisfy:

$$V_H \geq V_L + 2.5 \tag{5.6}$$

(the output range is made the same as the input range that will be discussed in the next paragraph). Offset and gain errors of the A/D converter are not taken into account in the foregoing formulas. Should these errors occur than the potentiometers can be adjusted so that the errors are compensated for. The operation of the circuit can be controlled by the following control lines:

* ENC: start conversion;
* /OC: read result of conversion from output latches;
* /WR: write address (A3..A0) of a multiplexer channel;
* /RS: reset multiplexer.

The timing requirements for these signals are shown in figure 5.7. A conversion is started by activating the ENC signal. This causes the ADC to sample and hold the signal at its input and convert this signal to a digital code. To determine the input channel that will be converted, the right address of the channel is written into the multiplexer's input latch using the /WR signal. When a conversion is completed, the output of the ADC automatically is clocked into the latches. This output code can be read from these latches

using the /OC signal. A conversion thus can be done with the following procedure:
1. write address into multiplexer latch;
2. write dummy word (activate the ENC signal) to ADC to start a new conversion;
3. read latches.

The time between 1 and 2 must be more than 450ns to allow the signals to settle. However, when more than one conversion is done consecutively, the next channel of the multiplexer can be chosen directly after a conversion start, since the ADC contains a sample and hold circuit. The latches can be read 800ns after the conversion is started.

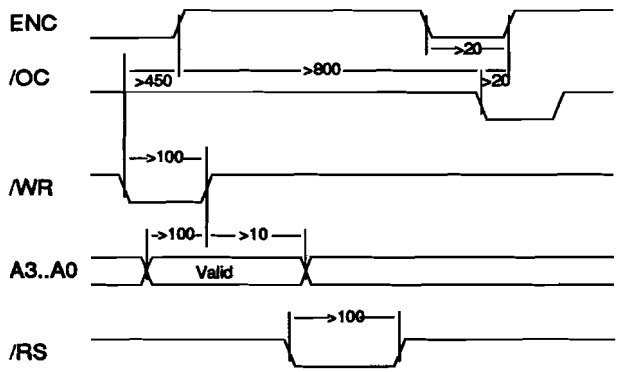


Fig. 5.7: Timing requirements for A/D circuit

The output of the A/D converter is a 12-bit two's complement code. The most negative number represents the lower boundary of the neural network output voltage (V_L), and the most positive number represents the upper boundary of the output voltage (V_H). The data formats of the multiplexer and the ADC code are shown in figure 5.8.

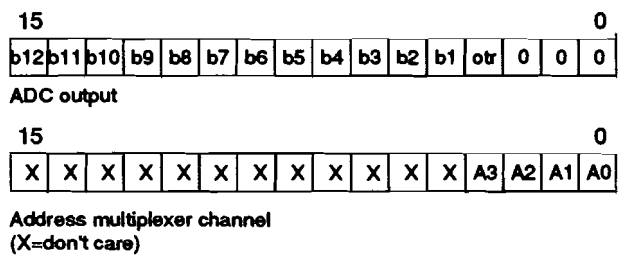


Fig. 5.8: Data formats A/D circuit

The outputs of the ADC are connected to the upper 12 bits of the 16 bit latch, so the most significant bit represents the sign bit. The out of range indicator of the converter is connected to bit 3 of the output latch, so it is always possible to detect an out of range

error (this error occurs whenever the input of the ADC is outside of the fixed range, $0 \leq V_{AD} \leq 5$). The lowest 3 bits (bit 2..0) of the output latch are always zero. The address that is used to choose one of the sixteen channels of the multiplexer must be present in the lower four bits of the 16-bit word. The required thirty-two analog input channels simply are formed by two identical circuits as shown in figure 5.6.

5.2.2 Digital to analog conversion

Analog inputs of the neural network all have to be stable at the same time. The digital to analog conversion therefore can be done in the two ways shown in the figures 5.9 and 5.10.

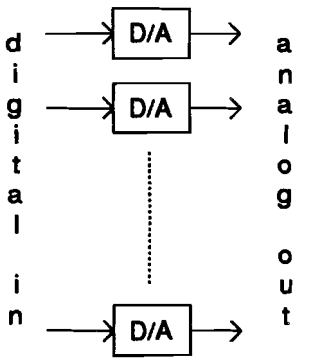


Fig. 5.9: Direct D/A conversion

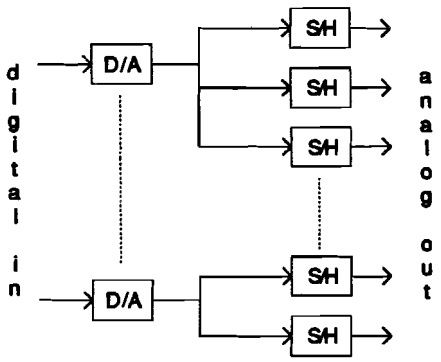


Fig. 5.10: Multiplexed D/A conversion

In the direct method all analog channels have their own D/A converter (DAC), while the multiplexed method increases the number of outputs with the help of sample/hold (S/H) devices. The use of S/H devices brings along several disadvantages. Although in this case a few fast D/A converters can be chosen costing not too much, the speed of the circuit is negatively influenced by the acquisition time of the S/H devices (this is the time that the device needs to track the signal again when changing from hold to sample mode). Cheap S/H devices have a large acquisition time, while the fast devices on the other hand result in an expensive board, because their prices are much higher (see appendix C for more data on some sample/hold devices).

Another problem that arises is the fact that each analog channel needs a sample/hold device. This results in a large printed circuit board, and this again results in an expensive design. The use of ICs with more than one sample/hold device does not solve the cost

problem since these ICs are very expensive.

Providing every analog channel its own D/A converter seems to be more attractive. With regard to the D/A converters the following must be noted:
1. the output range of the DAC must be adjustable between given lower and upper voltages;
2. if a current output DAC is used, the output has to be buffered by a fast opamp, that does not increase the conversion time significantly.

In case of a current output DAC, the buffer opamp can be used to adjust the output range, requiring no large additional circuitry. The best candidates to realize the analog output channels are the AD7568BP and the DAC8412EP, 12-bit converters from Analog Devices.

The AD7568BP contains eight current output converters, so the realization of thirty-two analog output channels requires only four of these ICs. Together with the buffer opamps (a suitable opamp is the AD713JN from Analog Devices) only twelve ICs are needed to realize the thirty-two voltage output channels. A serious disadvantage of this converter is the fact that digital data enters the chip serially. Complex circuitry is needed to control the input of digital data.

The DAC8412EP on the other hand is an IC containing four voltage output D/A converters. The output range is fully adjustable, with the only restriction that the range must be larger than 2.5 V. Although the price of this converter is higher than that of the AD7568BP, and the output range must be larger than 2.5 V, this converter has been chosen. The fact that the needed printed circuit board area is smaller, and the control logic is simpler outweighs the smaller costs of a circuit using the AD7568BP. In figure 5.11 a simple circuit, realizing four analog output channels is shown.

The output of the D/A converter is a voltage between V_{REFL} and V_{REFH}:

$$V_{out} = V_{REFL} + \frac{(V_{REFH} - V_{REFL}) * N}{4096} \tag{5.7}$$

with N the digital code in decimal. V_{REFL} and V_{REFH} can be set up with the potentiometers 3 and 4.

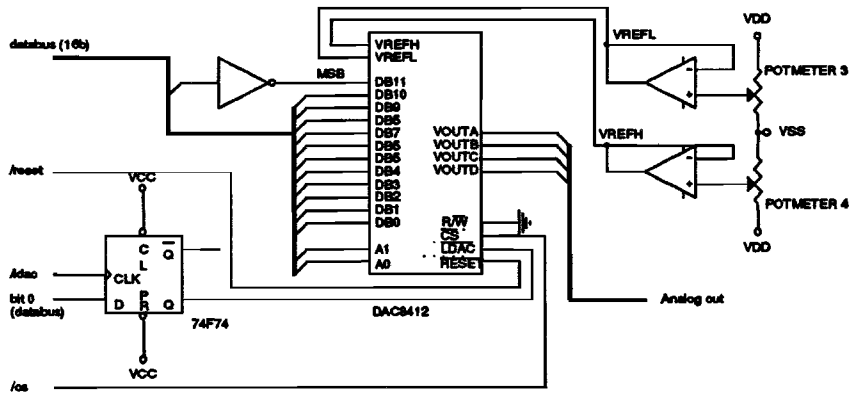


Fig. 5.11: Four analog output channels

The operation can be controlled by the following signals:

* /reset: reset all D/A converters to mid-scale
* /ldac: the converter can either be in update mode or in load mode. In the update mode the outputs of the converters are changed according to the codes present in the internal latches. In the load mode the contents of the internal latches can be changed, without changing the output voltages. /ldac is used to set the operating mode, on the rising edge of /ldac bit 0 of the databus is clocked into the flip-flop (1 for load mode and 0 for update mode).
* /cs: write a new code in the internal register of a DAC.

The timing requirements for these signals can be seen in figure 5.12. In figure 5.13 the data formats for the D/A circuit are depicted.

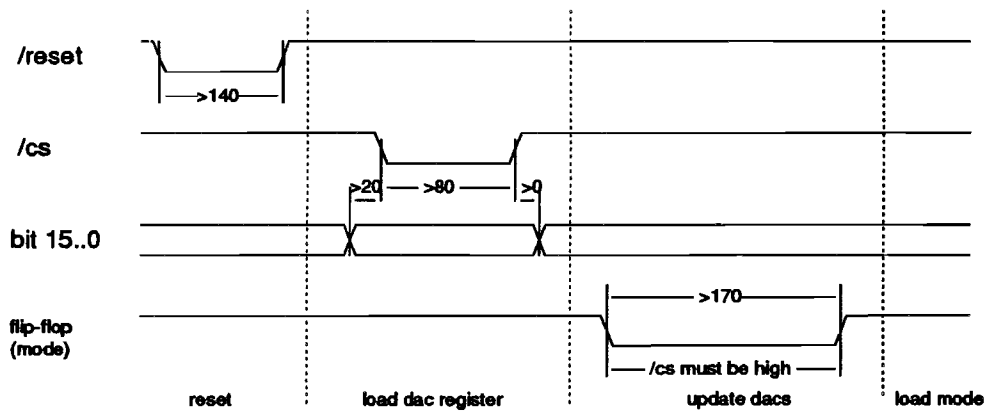


Fig. 5.12: Timing requirements for D/A circuit

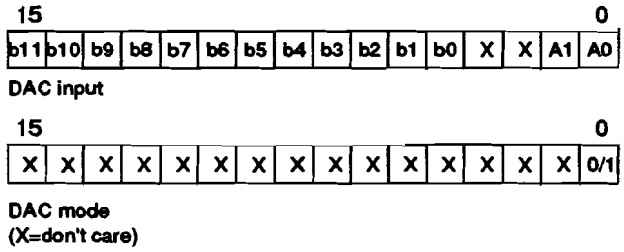


Fig. 5.13: Data formats D/A circuit

The conversion process now is done with the following procedure:
1. set the operating mode to load by writing a 1 into the mode flip-flop using the /ldac signal.
2. write the digital codes into the latches of the DACs. This is done by writing words according to the format shown in fig. 5.13 into the DAC's latches by using the /cs signal.
3. start the conversion by setting the converters in the update mode. This is done by writing a 0 into the mode flip-flop.

As can be seen in figure 5.13, the 12-bit two's complement code is supposed to occupy the highest twelve bits of the digital input word. The lowest two bits are used to address one of the four DACs that are present in each IC. The conversion time of all channels (all channels perform the conversion at the same time) is typically 6μs.

Extension of the circuit to more output channels can easily be done by adding converters. A thirty-two output channel circuit can be formed with eight converters connected in the way shown in figure 5.11.

5.2.3 Analog I/O circuit

A complete circuit with sixteen analog input and twenty analog output channels is shown in figure C.3 (Appendix C). It consists of the the subcircuits that are described in the foregoing. A circuit with thirty-two analog channels, just consists of two identical circuits as shown in figure C.3. The connectors shown in figure C.3 are used to connect the various signal to the bus interface and to the neural network. The power supplies are not shown in figure C.3. They must be realized externally since the power signals on the AT-bus and VL-bus connectors do not comply with the given specifations of the analog I/O circuit in appendix C. In Appendix C also the complete specifications of the analog I/O circuit are given. The realization of the given circuits on a printed circuit board will be discussed later. First, the interface to the bus will be discussed.

5.3 Interface to the AT-bus

The bus interface must provide all the signals required by the analog I/O circuit, the digital I/O circuit, and the neural network. This means that it should:
1. provide means to connect the databus to the components in the analog I/O circuit, the components in the digital I/O circuit, and the neural network circuitry;
2. provide means to address all components in the:
   - analog I/O circuit;
   - digital I/O circuit;
   - neural network (including possible RAM).

The first point can be done by using some bus drivers. Provision of (a part of) the computer's address bus to the neural network, also can be done with some bus drivers. Addressing all possible components requires some address decoding circuitry. A choice must be made whether to use ordinary memory addresses, or the special I/O addresses. This however will be discussed later, first the digital I/O circuit will be described. At the end of this paragraph the speed at which the complete interface can operate will be discussed.

5.3.1 Digital I/O

The digital I/O can easily be accomplished with some latches. In figure 5.14 two latches are shown, one configured as input and one configured as output.

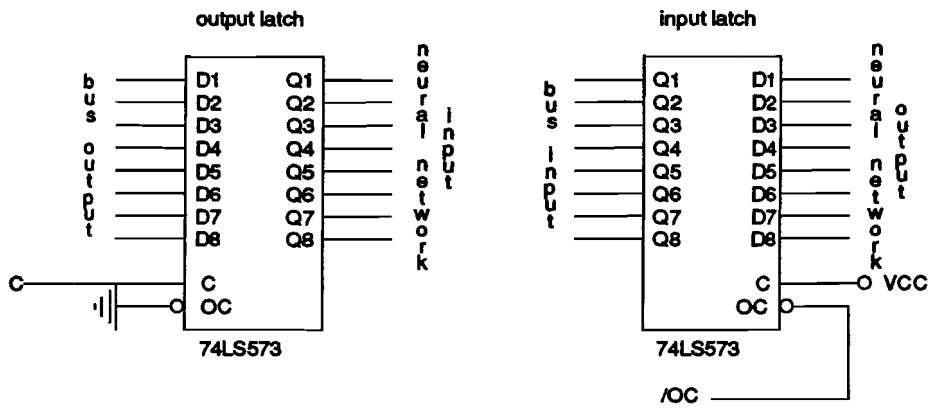


Fig. 5.14: Input and output latch

Data can be written in the output latches by making the C signal high. The outputs of these latches are always enabled. Digital data from the neural network can be read by the

computer by activating the /OC signal of the latch (making it low). The latches are included in the schemes shown in the figures C.3 and C.4 (the complete neural interface will contain twelve of the latches as shown in figure 5.14, six configured as input and six configured as output).

5.3.2 Bus interface circuit

As mentioned in chapter three, data should be transferred between the computer and the neural network as fast as possible. Also the interface must be as versatile as possible. The versatility is maintained by allowing all components to be addressed individually (especially the RAM of the neural network). There are two ways to address components:
1. by mapping them on ordinary memory;
2. by making use of the special I/O addresses. In this case there could even be made use of Direct Memory Access (DMA). When using DMA data is directly transported between I/O and memory without intervention of the CPU. This method is efficient when larger blocks of data have to be transferred.

Considering the fact that the interface has to be as versatile as possible, it is decided that all components (including RAM of the neural network) must be individually addressable in an efficient way. This excludes the use of DMA, since this method of transferring data brings along much overhead when used for single accesses to memory. However, in a memory-cached system, with pipelined execution of instructions as is the case in the 80386 and 80486 processor based systems, the use of ordinary (memory or I/O) buscycles should not be crucially slower (see [27] and [28] for data on DMA cycles).

As described in chapter four, the AT-bus supports several buscycles, with a predefined length. Since the speed of the bus directly influences the speed at which the neural interface operates, the length of the buscycles must be as small as possible. An examination of the timing diagrams of the buscycles (figures A.1 to A.6, appendix A), shows that the fastest data transfer rates can be achieved with zero waitstate 16-bit memory buscycles. Although the use of these cycles results in a somewhat larger circuitry for decoding and bus logic as with I/O buscycles, this method is chosen.

Now it is clear that the I/O will be mapped on memory, the actual bus interface can be designed. First of all, the memory addresses that the bus interface will respond to have to be chosen. There are two possibilities:
1. make use of the reserved address area;
2. make use of addresses above 1 MByte that are not in use by the system.

The first possibility is the most favorable. It results in simpler address decoding, since not all address lines have to be involved in the decoding (the /SMEMx signals that are only active for addresses in the lower 1 MByte memory can be used in the decoding circuitry), and future extension of the system memory is not prohibited by the fact that the addresses already are in use.

In figure C.4 (appendix C), a complete bus interface scheme is shown. The computer's data bus is buffered with 74LS245 bus transceivers. Address lines and other control signals are buffered with 74LS244 bus receivers. The circuit can be set up to one of two memory segments, namely the D or E segment (physical addresses D0000-DFFFF or E0000-EFFFF). A segment valid signal indicates that the segment is being addressed. This segment valid signal is also available on the neural network extension connector, a connector that also provides the lower 16 address lines, the /SMEMR, /SMEMW and the /RESET signals.

The analog and digital I/O circuits consist of about 25 components that must be addressed. This is done with help of four decoders, capable of addressing thirty-two addresses. The signals required by the circuit as shown in figure C.3 are connected to a connector in figure C.4. Two of these connectors make it possible to address all components on the analog I/O, consisting of two identical circuits as shown in figure C.3. The remaining 65,504 addresses can be used by the neural network, e.g. to address weights stored in off-chip RAM.

The use of zero waitstate 16-bit memory cycles requires the bus interface to return two signals to the bus logic on the mainboard during a buscycle, namely /MEMCS16 and /0WS. The timing of the /0WS signal is very critical. The signal must be returned within 10 ns after activation of the /MEMx signal. This is done by using the /MEMx signals, the segment valid signal (this is present before the /MEMx signals, since the addresses are earlier available) and with help of fast logic. The open collector nand gate 74F3038 (capable of driving a 30 Ω load resistance) is used to generate the /0WS signal.

The /MEMCS16 signal must be returned within 80 ns after the large address signals are valid. The easiest way to do this, is to use a 74F3038 connected to the large address bits 18 and 19. In this way all addresses on the bus above 768 K (segments C, D, E and F) are seen as 16-bit addresses, but this should not cause any problems in practice.
In appendix C, the exact physical addresses are shown of all the available analog and digital channels on the neural interface.

Care must be taken with the choice for the segment in which the neural interface will be installed. This segment may not be in use by another peripheral card, nor may it be used by a memory manager (the segment must be excluded for the memory manager). The reason for this is the fact that a memory manager allows DOS to make a call to an address in the reserved memory area, even if no physical memory is present. The memory manager takes care that the address is translated into an existing physical address that is outside the memory range that can be addressed by DOS. Should there be physical memory in the reserved area, e.g. the neural interface, then this memory will not be seen by DOS.

The reserved area also best can be excluded from the cacheable memory, so no problems such as those mentioned in chapter four, can arise when using cache memory.

5.3.3 speed of the neural interface

As stated before, the speed of the neural interface mainly will be determined by the length of the buscycles. A 16-bit zero waitstate buscycles can be completed in about 375ns. As described earlier a complete input cycle consists of the following actions:
1. choose input channel (write address to multiplexer);
2. start conversion (write dummy word to ADC);
3. read result of conversion.

All these actions require a buscycle, so the total process would last 3x375+800=1925ns. However, the total time needed for buscycles and conversion time can be shortened due to the possibility to have the cycles overlapping each other. If more than one conversion is performed consecutively, the next input channel can be chosen directly after the previous conversion is started. Also the buscycle that is needed to start the conversion overlaps with the actual conversion time. Further, the two present ADCs can operate in parallel, so the conversion time for two channels is the same as the time needed to convert a single channel. Estimates of the actual times needed for a complete input cycle are shown in table 5.1.

It must be noted that the CPU overhead (instruction and operand fetch from memory and execution of instruction) must be added to these times. These times however are of minor importance in a system that uses a memory cache and performs the instruction execution with help of a pipeline.

Table 5.1: Estimate of AT-bus input cycle times

number of input channels	time (µs)
1	1.55
2	2.675
16	18.75
32	36.75

An output cycle consists of the following actions:
1. write data to DAC;
2. set update mode;
3. start conversion;
4. set load mode.

All these cycles again require a buscycle. Actions 2,3,4 only have to be taken when the desired number of DACs has received new data. The conversion time is independent of the number of channels that have to be converted, since all channels are being converted at the same time. In table 5.2 estimates are given for the times needed to complete an output cycle. Again the CPU overhead has to be added to these times to get the exact time needed to complete the cycle, so in practice the performance will be somewhat worse.

Table 5.2: Estimate of AT-bus output cycle times

number of output channels	time (µs)
1	7.125
2	7.5
16	12.75
32	18.75

The time needed to update thirty-two input- and output channels is equal to 55.5 µs excluding CPU overhead. This means that the maximum processing rate of the interface is smaller than 18,000 vectors/s.

5.4 Interface to the VL-bus

The speed that can be achieved with the AT-bus interface is not very high. This, however is not surprisingly, since the AT-bus is a 16-bit bus, and the fastest buscycles still take 375 ns. The high-speed 32-bit VL-bus thus can be a very good alternative when trying to decrease the processing time of the neural interface. In the following a bus interface circuit for the VL-bus will be described. First, the digital I/O will be discussed, then again the address decoding circuitry will be described, and finally the speed at which the neural interface can operate will be discussed.

5.4.1 Digital I/O

The digital I/O does not have to be changed. The same circuit as shown in figure 5.14 can be used to provide digital inputs and outputs. Only now, data will be transferred with thirty-two bits at a time, instead of with sixteen bits.

5.4.2 Bus interface circuit

Although the data transfer rate is more than doubled when using the VL-bus instead of the AT-bus, the bus interface circuit will not be as small and simple. The length of the buscycles is not predefined and has to be determined by the peripheral card. The bus interface now will be designed for a 32-bit analog I/O circuit and neural network circuit. To save logic, only 32-bit accesses are allowed, so possible RAM of the neural network, also has to be accessed with thirty-two bits at a time. Since the fastest personal computer with a VESA local bus, available at the moment is a computer with the 80486DX2-66 processor, externally operating at 33 MHz and internally at 66 MHz, the bus interface will be designed for a 33 MHz bus.

A 32-bit analog I/O circuit can be formed by connecting two of the circuits as shown in figure C.3 to the 32-bit databus. The data formats for the circuit stay the same as shown in the figures 5.8 and 5.13, only now the 32-bit double word is made up of two 16-bit words. Again a choice must be made whether to use memory or I/O addresses. The difference in buscycle length of memory and I/O cycles no longer exists, since the cycles are not predefined. However, because of the fact that memory addresses are easier to deal with in software, the bus interface again will be memory mapped. To save logic it will be no longer possible to select the data segment, the circuit will respond to. Only the D segment will be used. Since all thirty-two address lines are available, more decoding logic is needed if the circuit is allowed to respond only to the addresses assigned to it. By

using only the lower 24 address bits, logic is saved, but the circuit will also respond to addresses above 16 M. So to guarantee correct operation of the circuit, no physical memory may be present in this range.

The bus can be used without the insertion of waitstates, but in this case it is possible to complete a buscycle in three clockcycles (about 100ns), and this again causes problems for the slower components in the analog I/O circuit, that require a minimum write pulse width of 100ns. So to overcome these problems, waitstates have to be inserted. The length of the buscycles now can be determined by the circuit shown in figure 5.15.

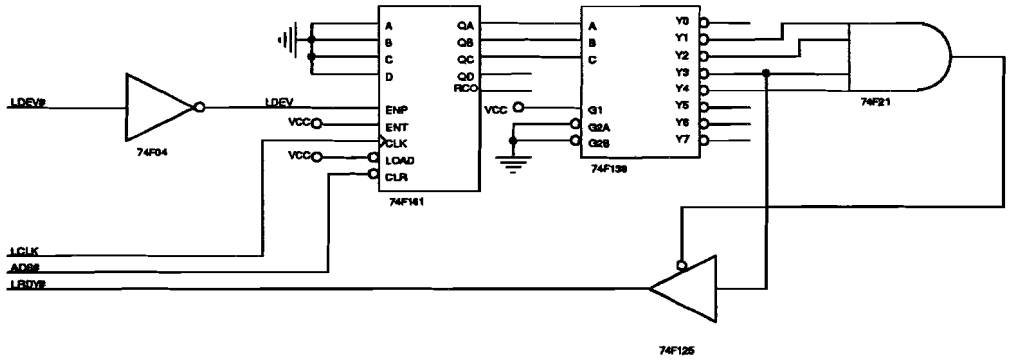


Fig. 5.15: Control of VL-bus cycle length

The begin of a transfer resets the 74F161 counter (that is only enabled when the circuit actually has to respond, with help of the LDEV# signal). On each positive edge of LCLK, the counter enters a new state. On the fourth positive edge the cycle is ended by disabling the 74F125 output buffer again (of the LRDY# signal). In figure 5.16 a timing diagram of the circuit is shown.

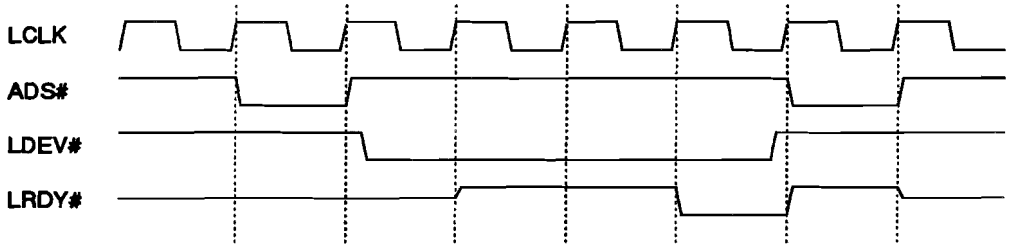


Fig. 5.16: VL-bus cycle length timing

No difference is made between read and write cycles, so both will last about 165 ns. In case of a read cycle, the return of the RDYRTN# signal is not awaited to stop driving the busses, since this signal is returned in the same cycle as the LRDY# signal in a 33 MHz system.

The address decoding circuit is made up of some address decoders (74F138 and 74F538). This part of the circuit does not differ significantly from the AT-bus circuit.
A complete circuit for the VL-bus interface is shown in figure C.6 (appendix C), a detailed timing diagram of this circuit is shown in figure C.7. The physical addresses are also given in appendix C.

Although the idea to realize the circuit in this way seems to be good, the timing of the circuit cannot meet the required specifications as described in chapter four when fast logic components are used. The LDEV# signal still can be returned in time with the circuit shown in figure C.6, but the LRDY# signal definitely cannot be returned in time. The maximum delay with regard to the positive edge of LCLK namely is 10ns (see figure B.5). In the timing diagram (figure C.7) it can be seen that the actual time is about 25 ns. This is caused by the fact that the signals must pass at least five layers of logic, all with delay times greater than 4 ns. The specification would allow only two layers of logic. It seems that this problem only can be solved by integrating the circuit on a single VLSI chip, e.g. a programmable logic device. This also would greatly reduce the area the circuit occupies.

However, if the mentioned timing problem is solved, and a correct circuit to define the length of a buscycle is realized, it must be possible to design a VL-bus interface circuit based on the circuit shown in figure C.5.

Further it must be noted that the circuit is designed under the assumption that the address and data lines remain valid until the LRDY# signal is returned by the LBT. The timing diagrams are not completely clear about this, and to be sure it would be best to check this in practice.

5.4.3 Speed of the neural interface

The speed of a circuit that is designed in the way described above will be much higher than in the case of the AT-bus circuit. A complete buscycle now only lasts 165 ns. The input cycle now consists of the following actions:
1. choose two input channels (write 32-bit double word to multiplexers);
2. insert delay to allow signals to settle;
3. start conversion (write dummy word to ADCs);
4. read result after conversion is completed.

Action 2 is only needed when only two channels have to be converted. In the case of the AT-bus, no additional delay had to be inserted because the time between two consecutive

bus cycles was long enough to allow the signals to settle. In case of the VL-bus, this time is not long enough anymore, and the processor has to wait before the conversion can be started (about 300 ns). When more than two channels have to be converted, the new channels can be chosen directly after a conversion is started, since the ADCs sample and hold the input voltages. Estimate of conversion times are given in table 5.3. The CPU overhead again must be added to get the actual times.

Table 5.3: Estimate of VL-bus input cycle times

number of input channels	time (µs)
2	1.43
16	8.43
32	16.43

The output cycle actions still are the same as in the case of the AT-bus interface:
1. write data to two DACs;
2. setup update mode;
3. start conversion;
4. set load mode.

All these actions require a buscycle. Actions 2,3 and 4 only have to be taken when the desired number of DACs has received new data. In table 5.4 estimates are given for the times needed to complete an output cycle.

Table 5.4: Estimate of VL-bus output cycle times

number of output channels	time (µs)
2	6.5
16	7.65
32	8.97

Thirty-two input- and output channels now can be updated in about 25 µs excluding CPU overhead. The maximum processing rate thus will be smaller than 40,000 vectors/s.

5.5 Realization of a printed circuit board

The circuits eventually have to be realized on a printed circuit board that can be inserted in a slot of the personal computer. The neural network then can be connected to this board with help of some cables and connectors. However, the size of this printed circuit board greatly influences the total costs of the interface. During the design it is already taken into account that the circuits have to be as small as possible. Attempts to realize layouts of the circuits shown in appendix C, failed in an early stage. It appears that much experience and time is needed to develop a layout that is as small as possible. This the reason why no layout will be made. The design of a layout best can be farmed out to people experienced at these matters. Yet, there are some guidelines that have to be reviewed when implementing the circuits on a printed circuit board. The guidelines for the different parts, analog I/O, digital I/O and bus interface circuit will be discussed separately.

It must be noted that all circuits are designed on basis of the manufacturer's specifications (data sheets of used components). To ensure that the timing specifications are met, the circuits should be tested in practice before being implemented on a printed circuit board.

5.5.1 Analog I/O PCB

The analog components in the analog I/O circuit are very sensitive to good grounding. A ground plane is highly recommended for this circuit. Also the ground references of the AD1671 ADC should be star connected. Since the power supplies available on the AT-bus connector have a too great tolerance, external power supplies must be used. Adequate power supply bypassing is required, the capacitors for this purpose are shown in the schemes in appendix C. The fact that in figure C.3 only a 16-channel analog I/O is shown is no coincidence. It is namely possible to realize two identical PCBs of this circuit, costing less than one PCB that contains the complete circuit. Especially with respect to the fact that the PCB must have more than one layer, this can reduce the costs significantly. Two 10x10 cm cards (with four layers) cost about fl. 1,300, whereas one 20x10 cm card costs about fl. 1,800. However, this way of implementing the circuit can only be done in connection with the AT-bus interface. The AT-bus circuit, namely can be realized on a separate PCB, providing connectors to connect both the analog I/O circuits and the neural network (the cables between the connectors should be as short as possible however). The bus interface circuit of figure C.4 is designed under this assumption. When interfacing to

the faster VL-bus this method probably cannot be chosen. Problems will arise when trying to connect the 33 MHz busses to other circuits with help of cables and connectors.

5.5.2 At-bus interface PCB

The circuit shown in figure C.4 can be realized on a separate PCB. The complexity of this circuit is not very high, and it should be possible to implement the circuit on a 10x16 cm large PCB. Such a card would cost about fl. 350 (not including the components). Also this card can be connected directly with the AT-bus, no external power supplies are required. Only care has to be taken when dealing with the fast logic components. All unused inputs of these components, even those on unused gates, should be tied to a voltage source of relatively low impedance.

5.5.3 VL-bus interface PCB

Since the circuit for the VL-bus interface of figure C.6 does not operate correctly, this circuit will not have to be implemented on a printed circuit board. However, if a VL-bus interface circuit that does meet the specifications is implemented, the result probably will be an expensive board. This is caused by the fact that the circuit operates at 33 MHz, bringing along several restrictions when designing the layout. Further the following guidelines must be taken into account:
- the distance between a line from the VL-bus connector to a component may be no more than two inches;
- power must be drawn equally from the power lines;
- the signal impedance on each trace should be less than 50 Ω. This impedance can be calculated with formula 4.2.

The realization of separate analog I/O cards also will be problematic, since the busses that connect the cards operate at 33 MHz.

Considering the foregoing remarks it will be very unlikely that a PCB with a VL-bus interface can be realized costing less than fl. 2,000.

5.6 Costs of the neural interface

The costs of the neural interface mainly will be determined by the costs of the printed circuit board. As mentioned before, the design of two separate analog I/O cards and one bus interface card, will result in a price of fl. 1,650 (providing that the analog I/O circuit can be implemented on a 10x10cm PCB). The components needed to realize the analog I/O circuit cost about fl. 2,200 and the components for the bus interface circuits cost about fl. 200. So the complete neural interface for the AT-bus will cost about fl. 4,000. However, this will be the price for a first version of the interface. Should any errors occur and a redesign would be needed, the costs will be much higher.

Interfacing to the VL-bus probably will be more expensive. The analog I/O part will cost the same as in the case of the AT-bus interface. The components for the VL-bus interface circuit (as mentioned earlier this interface must be made using programmable devices or dedicated VLSI chips) will be more expensive. Also the PCB will cost more than in the case of the AT-bus interface. It is very well possible that a first version of a VL-bus neural interface will cost fl. 6,000. Should a redesign be necessary, then the costs are very likely to be larger than fl. 8,000.

5.7 Software for the neural interface

The neural interface must be operated with software. In the following, basic input and output routines will be discussed for the AT-bus interface. In first instance the routines will be written in the C programming language. To show how the basic routines can be used, an example will be discussed, in which a neural network chip-set with the back-propagation implemented in hardware, will be controlled by the neural interface. This example however is only a rough indication since no exact information on the hardware is available. At this moment it is not possible to write complete, correctly functioning, programs. First of all the data formats of variables that are used in conjunction with the neural interface will be discussed.

5.7.1 Data formats

The data formats of the neural interface have been discussed in paragraph 5.2. The 12-bit two's complement code that is written to and read from the DACs and ADCs must occupy the upper twelve bits of a 16-bit word. Integer values to be sent to a DAC must have the lower four bits set to zero. In this way the sign of the code is preserved, and the 16-bit word can be dealt with as an ordinary integer (in C integers have a length of 16 bits), although the actual value of the integer is the 12-bit code used by the neural interface multiplied by sixteen. Also it still is possible to add the address of the output channel in a single DAC IC, without changing the 12-bit code that is sent to this DAC.

When processing data of the neural network, operations can be performed on these integers without many problems. Only in case of multiplication and division care must be taken. When performing a multiplication, the result must be divided by 16, and when performing a division, the result must be multiplied by 16 to maintain the special format. However, should the 12-bit code have occupied the lower 12 bits of a 16-bit integer, a conversion would be needed for every operation (the sign bit of the code would have to be present in the most significant bit, bit 16, instead of in bit 12), and this would be more inefficient than scaling the results of a multiplication or a division.

If floating point operations are needed, the special integer values can be converted to floating point values. This can be done with the following functions (see also Appendix D):

float integer_to_float(int integer, float lower, float upper)

This function converts an integer according to the format of figure 5.13 to a floating point

variable representing the actual voltage (between the boundaries lower and upper).

int float_to_integer(float floating, float lower, float upper)
This function converts a floating point value representing a voltage (between the boundaries lower and upper) to an integer according to the format shown in figure 5.13.

5.7.2 Basic input and output routines

Since the complete neural interface is memory mapped, all components can be accessed with pointers in C (far pointers must be used, since the D and E segment will be outside the code and data segment that are in use by a C program). The base addresses of these pointers can be defined by adding the following lines to a C program (the interface is chosen to occupy the D segment as an example):

```
#define   dac_base      0xD0000 /* base address of D/A converter ICs */
#define   dac_mode      0xD0014 /* base address of mode flip-flop */
#define   mux_base      0xD0016 /* bsae address of two multiplexers */
#define   adc_base      0xD0020 /* base address of two A/D converters */
#define   dig_base      0xD0024 /* base address of three 16-bit digital latches */
```

Before the input latches of the D/A converters can be loaded the mode flip-flop must be reset (all other components are automatically reset by the system reset). This can be done with:

```
long      far *modeptr;

modeptr=dac_mode;
*modeptr=0; /* write 0 into mode flip-flop */
```

The following basic input and output functions are available (see Appendix D):

void load_single_dac (int channel, int value)
This function loads a single value into the latch of one of the forty output channels. The parameter value must contain a 12-bits word (two's complement) in the upper twelve bits, the lower four bits must be zero (the addresses of the right output channel in a single IC are added by the function).
Example of a function call:
load_single_dac(23,0x30).

void load_all_dacs (int *value, int number)
A number of channels (up to forty) that are in use as input by the neural network can be loaded with this function. The values that will be loaded are passed to the function by a pointer that indicates the first element of an array of integers (the length of this array must be equal to number). The lower four bits of the integers again must be zero (same as with the previous function).
Example:
load_all_dacs(&v[0], 32).

void update_dacs (void)
The outputs of the DACs are changed by calling update_dacs. Note that the loading of new values in the DACs and the update of the output of these DACs can be done independently. Output channels that are not changed remain unchanged after the update.

int read_single_adc (int channel)
This function returns the result of a conversion of a single input channel (one of thirty-two). The format of the result is in agreement with figure 5.8.
Example:
i=read_single_adc(17).

void read_all_adcs (int *value, int number)
A number of input channels (this number must be even and smaller than or equal to thirty-two) can be read with this function. The results are stored in an array of integers, of which a pointer to the first element must be passed to the function.
Example:
read_all_adcs(&v[0],32).

void process_all_dacs_adcs (int *dacvalues, int *adcvalues)
This function performs a complete input and output of thirty-two analog channels. It is a combination of the functions load_all_dacs, update_dacs, and read_all_adcs. The parameters dacvalues and adcvalues are pointers to the first elements of two arrays of thirty-two integers, one containing data to be output, and one in which the input results will be stored.
Example:
process_all_dacs_adcs(&d[0],&a[0]).

void write_digital (int number, int value)
A 16-bit word is written into one of the three digital output latches with write_digital. Number must be 0, 1, or 2.
Example:
write_digital(2,0x34A8).

int read_digital (int number)
The digital input latches are read with read_digital, number indicating one of the three 16-bit input latches.
Example:
x=read_digital(1).

All the functions have been written in the C programming language. It may be possible to speed up the programs by replacing some functions by assembly code. This can be done by optimizing the assembly code after compilation of the functions. In the following these functions will be used in an example.

5.7.3 Example: Back-propagation program

Since no neural networks chips are available yet, the functions will be used to control an imaginary neural network system, as shown in figure 5.17.

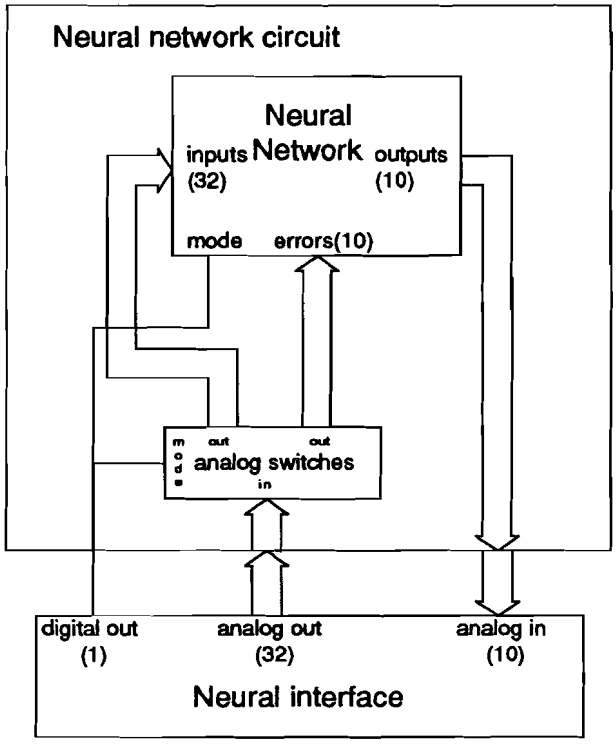


Fig. 5.17: Imaginary neural network system

The neural network is assumed to be made up of chips with the back-propagation algorithm (as described in chapter two) implemented in hardware. The system has thirty-two analog inputs (with a certain range), ten analog outputs (also with a certain range), ten error inputs (new weights are determined with help of these errors), and one digital mode bit to set the operation mode. This mode bit sets the analog switches in the right direction and determines whether the neural network processes input data, or updates weights (normal mode is indicated with bit=0 and update mode with bit=1). The mode bit is assumed to be connected to bit 3 of digital latch 0. The analog outputs of the neural network are connected to the first 10 analog input channels of the neural interface, and the 10 error inputs are connected to the first 10 analog output channels of the neural interface.

The testpatterns that are used to learn the network are assumed to be present in an array of 100 testpatterns, each consisting of 32 integers in the right format (lower four bits zero): testpatterns[100][32]. The correct outputs for these inputs are assumed to be present in an other array: correct_outputs[100][10], in the same format as the inputs. The system now can be controlled in the following way:

1. present input to network;
2. wait until inputs are processed by network and outputs are valid;
3. read outputs;
4. determine errors;
5. set update mode;
6. present errors to network;
7. wait until all weights are updated;
8. set normal mode again;
9. go back to 1. if error is not small enough yet.

If the difference between the actual output values and the desired values is calculated with ordinary integer subtraction and the error is determined using floating point values according to formula (2.4), these actions can be encoded in software in the following way (the action that a program line is part of is shown next to the program line):

```
while (error/2>1e-3)
{
        for (i=0;i<100;i++)
        {
                load_all_dacs(&testpatterns[i][0],32);                  1
                update_dacs;                                            1
                for (j=0;j<1000;j++);                                   2
                read_all_adcs(&test_output[0],10);                      3
                for (j=0;j<10;j++)
                {
                        difference[j]=correct_output[i][j]-test_output[j];  4
                }
                write_digital(0,0x04);                                  5
                load_all_dacs(&difference[0],10);                       6
                update_dacs;                                            6
                for (j=0;j<1000;j++);                                   7
                write_digital(0,0x00);                                  8
                for (j=0;j<10;j++)
                {
                        float_dif=integer_to_float(difference[j],-2.5,2.5);  9
                        error+=float_dif^2;                             9
                }
                if (error/2<1e-3) i=100;                                9
        }
}
```

Intermediate results are stored in the arrays test_output[10] (used to store the outputs of the neural network), and difference[10] (used to store the difference between the desired output and the actual output of the network). The voltage range of the neural network in the example is chosen to be: $-2.5 \leq V_{NN} \leq 2.5$. The program is stopped if the error becomes smaller than 1.10^{-3}. Two for-loops are used to insert delays to allow the neural network to process data. The number of times these loops are executed are a measure for this processing time.

6 Conclusions

The design of a versatile interface between neural network chips and a personal computer is not as easy as it might look in first instance. Many problems are encountered, especially if all given specifications have to be met. The use of commercially available interfaces is no solution if not much money can be spent. A fast and reasonably versatile interface costs more than fl. 10,000. However, such an interface is guaranteed to work and can be used immediately without having to write basic software routines. The design of an own interface can be done at a lower price but the performance will be worse (although the versatility can be greater). Of the two discussed designs, one for the AT-bus and one for the VL-bus, the AT-bus interface seems to be the least expensive, while still a reasonable performance can be attained. Both designs however do not meet all the specifications given in chapter three. Especially the required processing speed can not be attained at the allowed price of fl. 2,500.

With respect to the AT-bus interface the following remarks can be made:

- the interface is versatile, it contains 32 analog voltage inputs, 40 analog voltage outputs, 48 digital inputs, 48 digital outputs, fully adjustable voltage ranges;
- the maximum processing rate is 18,000 vectors/s (32-channel analog vectors);
- a first version of the interface will cost about fl. 4,000;
- a printed circuit board still must be developed (the circuit also must be tested in practice before realizing the printed circuit board) if the circuit actually is to be used;
- software can be written in a high level programming language like C without any problems;
- a 80486-type computer is highly recommended to run the software on, especially when the update of weights has to be done by this computer;

With respect to the VL-bus interface the following can be said:

- it is impossible to design such an interface without using fast programmable devices or dedicated VLSI chips;
- the price of such an interface very likely will be higher than fl. 6,000;
- the maximum processing speed of an interface that uses the same analog I/O circuit as the AT-bus interface is 40,000 vectors/s (32-channel analog

vectors);

- since the VL-bus is restricted to frequencies below 40 MHz, the fastest personal computer with a VESA local bus available at the moment is a computer based upon the 80486DX2-66 processor, internally operating at 66 MHz and externally operating at 33 MHz (so the local bus also operates at 33 MHz);

With respect to the software for the neural interface the following remarks can be made:

- all software still must be tested when real hardware is available;
- it may be possible to speed up the programs by writing routines in assembly language and call these routines in the C-program;
- the development of complete programs for the control of neural network chips, cannot be done before exact information is available about the hardware of the neural network circuitry.

7 Recommendations

At the moment, the best option for a neural network interface seems to be a commercially available interface. Although this may be not as versatile as an own interface, the functioning of such an interface is guaranteed and it can be used immediately. The processing speed can even be higher if it is no longer required to use a personal computer. The interface discussed in [15] than is a very good alternative.

Since the circuits presented in this thesis still must be tested and layouts for a printed circuit board have to be made, it can not be guaranteed that the total costs of the interface will be not more than fl. 5,000 and none of the circuits meets all the specifications given in chapter 3, a commercially available interface is a very good alternative to test realized VLSI chips containing neural networks.

If the chips really have to be used at their normal processing speed, the design of an own interface (possibly based on the circuits discussed in this thesis) can be reconsidered. However, a budget of over fl. 10,000 is highly recommended in that case.

Bibliography

[1] Bailes, L. et al.
Memory management and multitasking beyond 640k.
Blue Ridge Summit : Windcrest, 1992.

[2] Bruin, P.P.F.M.
A weight perturbation neural net chip set.
Eindhoven: Eindhoven University of Technology, Electronic Circuit Design Group, 1993.

[3] Brumm, P. et al.
80486 programming.
Blue Ridge Summit: Windcrest, 1991.

[4] Claasen-Vujcic, T.
Implementation of a multi-layer perceptron using pulse-stream techniques.
Eindhoven: Eindhoven University of Technology, Electronic Circuit Design Group, 1993.

[5] Cram, R.M.
Microcomputer busses.
San Diego: Academic Press, 1991.

[6] Duranton, M. et al.
A digital VLSI module for neural networks.
In: Proceedings of 'nEuro',
Paris, 6-8 July 1988.

[7] Fang, W., et al.
A VLSI neural processor for image data compression using self-organization networks.
IEEE Transactions on neural networks, vol. 3(1992), no. 3, pp. 506-517.

[8] Forney, J.
MS-DOS beyond 640k working with extended and expanded memory.
Blue Ridge Summit: Windcrest, 1989.

[9] Graf, H.P. et al.
Reconfigurable neural net chip with 32K connections.
In: Advances in neural information processing systems 3 (1991),
Ed. by D.S. Touretzky and R. Lippman,
San Mateo, CA: Morgan Kaufmann.

[10] Graf, H.P.
A neural-net board system for machine vision applications.
Proceedings of international conference on neural networks, 1991, pp. I-481 - I-486.

[11] Hertz, J., Krogh, A. and Palmer, R.G.
Introduction to the theory of neural computation.
Redwood City: Addison-Wesley Publishing Company, 1991.

[12] Holler, M. et al.
An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses.
In: The proceedings of the international annual conference on neural networks, Vol. II, Washington D.C., June 1989, pp. 191-196.

[13] Intel 80170NX electrically trainable analog neural network.
Intel, June 1991.
Order number: 290408-002.

[14] Jabri, M. et al.
Weight perturbation: an optimal architecture and learning technique for analog VLSI feedforward and recurrent multi-layer networks.
IEEE Transactions on neural networks, vol. 3(1992), no. 1, pp.154-157.

[15] Jiggle's user manual.
Sydney: University of Sydney, Department of Electrical Engineering, Systems Engineering and Design Laboratory, 1993.

[16] Kate, R. ten
A study of the weight perturbation algorithm in neural networks.
Eindhoven: Eindhoven University of Technology, Electronic Circuit Design Group, 1993.

[17] Leventhal, L.A.
Lance Leventhal's 80386 programming guide.
Toronto: Bantam Books, 1987.

[18] Lippmann, R.P.
An introduction to computing with neural nets.
IEEE ASSP Magazine, April 1987, pp. 4-22.

[19] Marshall, T.
Fast transit. Slow slots? VL-Bus, PCI and Quickring will break system bottlenecks without walloping your wallet.
Byte, October 1992, pp. 122-136.

[20] Mauduit, N. et al.
Lneuro 1.0: A piece of hardware lego for building neural network systems.
IEEE Transactions on neural networks, vol. 3(1992), no. 3, pp. 414-422.

[21] Melton, S.M. et al.
The TInMANN VLSI chip.
IEEE Transactions on neural networks, vol. 3(1992), no. 3, pp. 375-384.

[22] Mumford, L. et al.
The mod 2 neurocomputer system design.
IEEE Transactions on neural networks, vol. 3(1992), no. 3, pp. 423-433.

[23] Pétin, Y.A.
Implementation of a multi-layer perceptron including back propagation training algorithm.
Eindhoven: Eindhoven University of Technology, Electronic Circuit Design Group, 1993.

[24] Säckinger, E. et al.
Application of the ANNA neural network chip to high-speed character recognition.
IEEE Transactions on neural networks, vol. 3(1992), no. 3, pp. 498-505.

[25] Satyanarayana, S. et al.
A reconfigurable VLSI neural network.
IEEE Journal of solid-state circuits, vol. 27(1992), no. 1, pp. 67-81.

[26] Schnurer, G.
Local-Matadoren. Local-Bus-Systeme im Überblick.
C'T Magazine, Heft 9, 1992, pp. 99-108.

[27] Stiller, A.
AT-Bus, Die Busspezifikation des PC/AT gemäß IEEE P996.
C'T Magazine, Heft 11, 1991, pp.336-342.

[28] Stiller, A.
AT-Bus Timing, Timing-Diagramme für die Buszyklen gemäß IEEE P996.
C'T Magazine, Heft 12, 1991, pp. 313-318.

[29] The Engineering Staff of Analog Devices, Inc., ed. by Sheingold, D.H.
Analog-Digital conversion handbook.
Englewood Cliffs: Prentice-Hall, 1986.

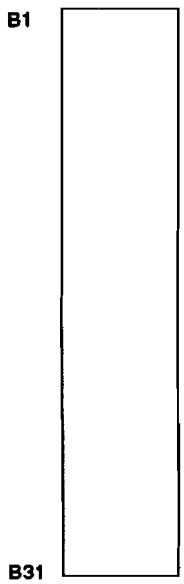
[30] TMS320C30 PC processor board. Technical reference manual.
Loughborough: Loughborough Sound Images, 1991.

[31] VESA Local Bus Proposal: general design overview.
San Jose, CA: VESA Video Electronic Standards Association, 1992.

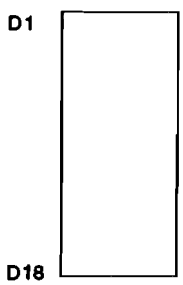
[32] VESA VL-Bus local bus standard, revision 1.0.
San Jose, CA: VESA Video Electronic Standards Association, 1992.

[33] Yasunaga, M. et al.
A self-learning neural network composed of 1152 digital neurons in wafer-scale LSIs
IEEE International joint conference on neural networks, vol. 3(1991), pp. 1844-1849.

Appendix A. AT-bus data



Signal	I/O	Pin	Pin	I/O	Signal
GND		b1	a1	I	/IOCHCK
RESDRV	O	b2	a2	I/O	SD7
+5 V		b3	a3	I/O	SD6
IRQ2	I	b4	a4	I/O	SD5
-5 V		b5	a5	I/O	SD4
DRQ2	I	b6	a6	I/O	SD3
-12 V		b7	a7	I/O	SD2
/0WS	I	b8	a8	I/O	SD1
+12 V		b9	a9	I/O	SD0
GND		b10	a10	I	IOCHRDY
/SMEMW	O	b11	a11	O	AEN
/SMEMR	O	b12	a12	I/O	SA19
/IOW	I/O	b13	a13	I/O	SA18
/IOR	I/O	b14	a14	I/O	SA17
/DACK3	O	b15	a15	I/O	SA16
DRQ3	I	b16	a16	I/O	SA15
/DACK1	O	b17	a17	I/O	SA14
DRQ1	I	b18	a18	I/O	SA13
/REF	I/O	b19	a19	I/O	SA12
CLOCK	O	b20	a20	I/O	SA11
IRQ7	I	b21	a21	I/O	SA10
IRQ6	I	b22	a22	I/O	SA9
IRQ5	I	b23	a23	I/O	SA8
IRQ4	I	b24	a24	I/O	SA7
IRQ3	I	b25	a25	I/O	SA6
/DACK2	O	b26	a26	I/O	SA5
TC	O	b27	a27	I/O	SA4
BALE	O	b28	a28	I/O	SA3
+5 V		b29	a29	I/O	SA2
OSC	O	b30	a30	I/O	SA1
GND		b31	a31	I/O	SA0



A/C: component side
B/D: solder side

Signal	I/O	Pin	Pin	I/O	Signal
/MEMCS16	O (/I)	d1	c1	I/O	/SBHE
/IOCS16	O (/I)	d2	c2	I/O	LA23
/IRQ10	I	d3	c3	I/O	LA22
/IRQ11	I	d4	c4	I/O	LA21
/IRQ12	I	d5	c5	I/O	LA20
/IRQ13	I	d6	c6	I/O	LA19
/IRQ14	I	d7	c7	I/O	LA18
/DACK0	O	d8	c8	I/O	LA17
/DRQ0	I	d9	c9	I/O	/MEMR
/DACK5	O	d10	c10	I/O	/MEMW
/DRQ5	I	d11	c11	I/O	SD8
/DACK6	O	d12	c12	I/O	SD9
/DRQ6	I	d13	c13	I/O	SD10
/DACK7	O	d14	c14	I/O	SD11
/DRQ7	I	d15	c15	I/O	SD12
+5 v		d16	c16	I/O	SD13
/MASTER	I	d17	c17	I/O	SD14
GND		d18	c18	I/O	SD15

Fig. A.1: Pin identification and signals of AT-bus

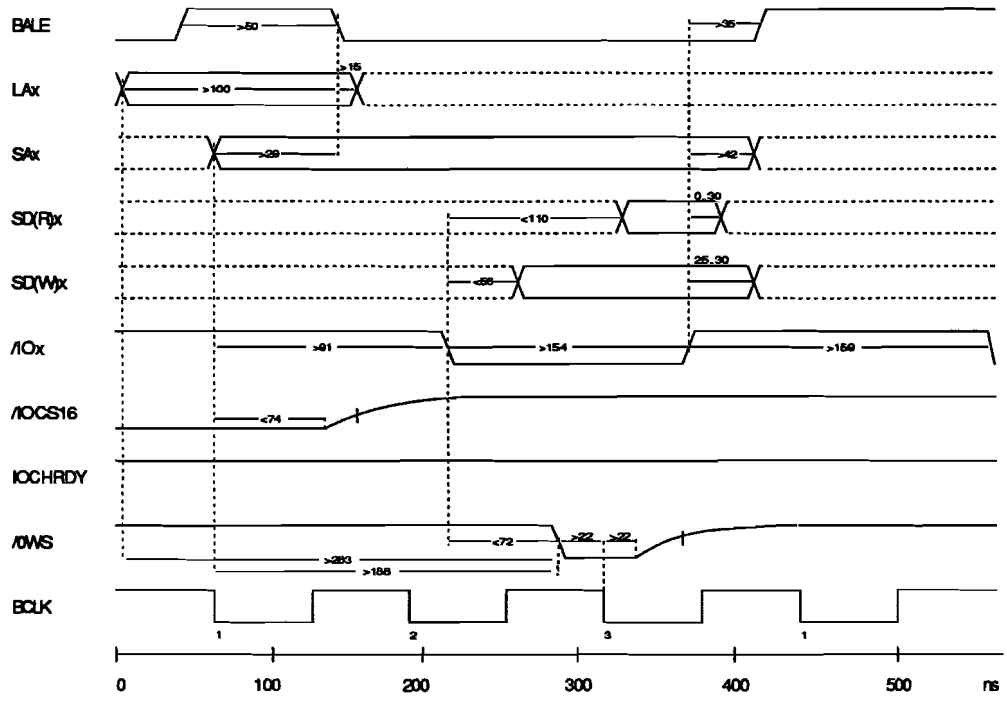


Fig. A.2: 8-bit IOx zero waitstate cycle

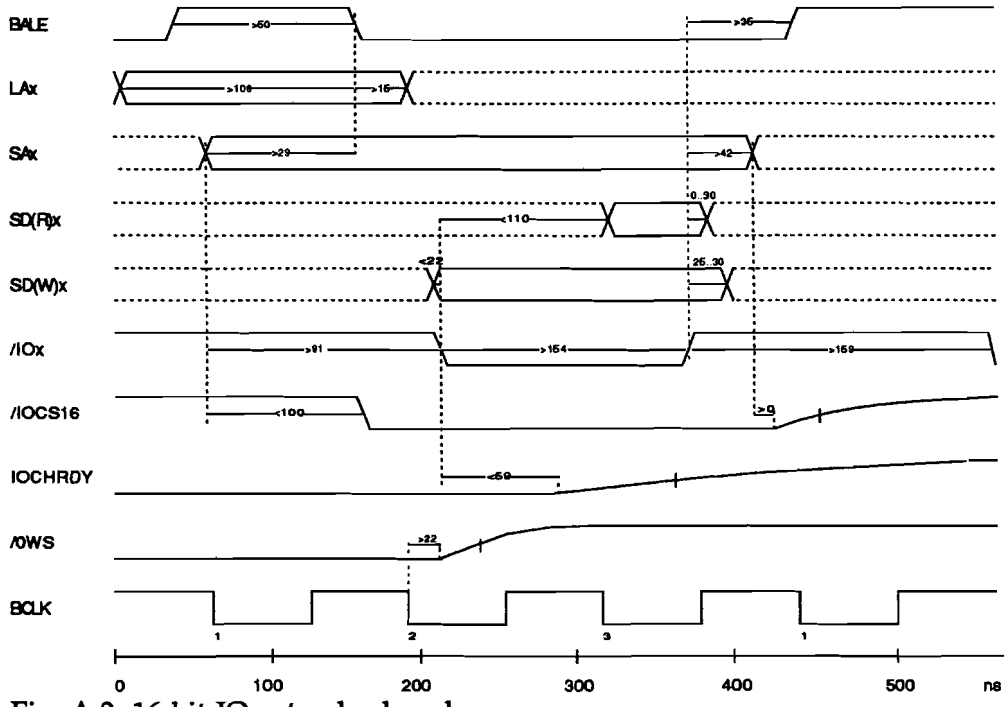


Fig. A.3: 16-bit IOx standard cycle

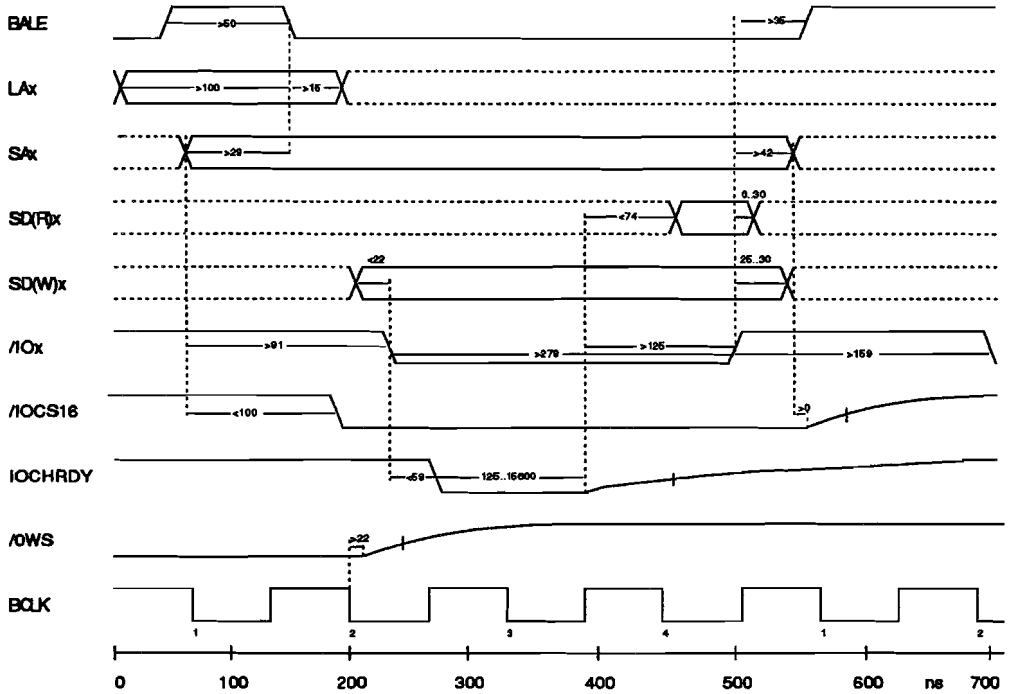


Fig. A.4: 16-bit IOx ready cycle

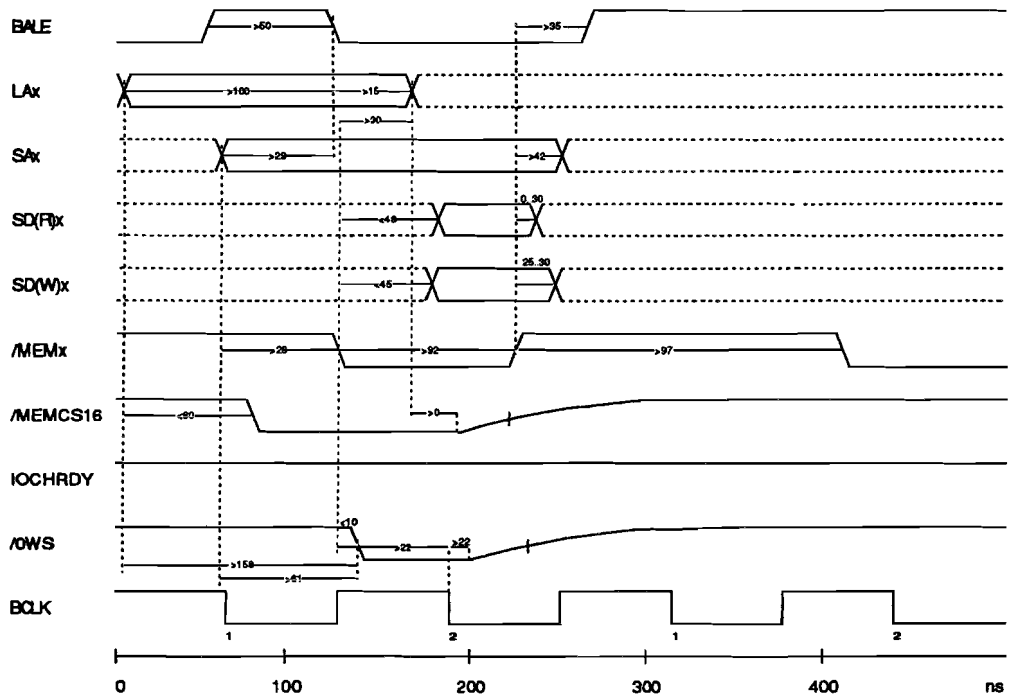


Fig. A.5: 16-bit MEMx zero waitstate cycle

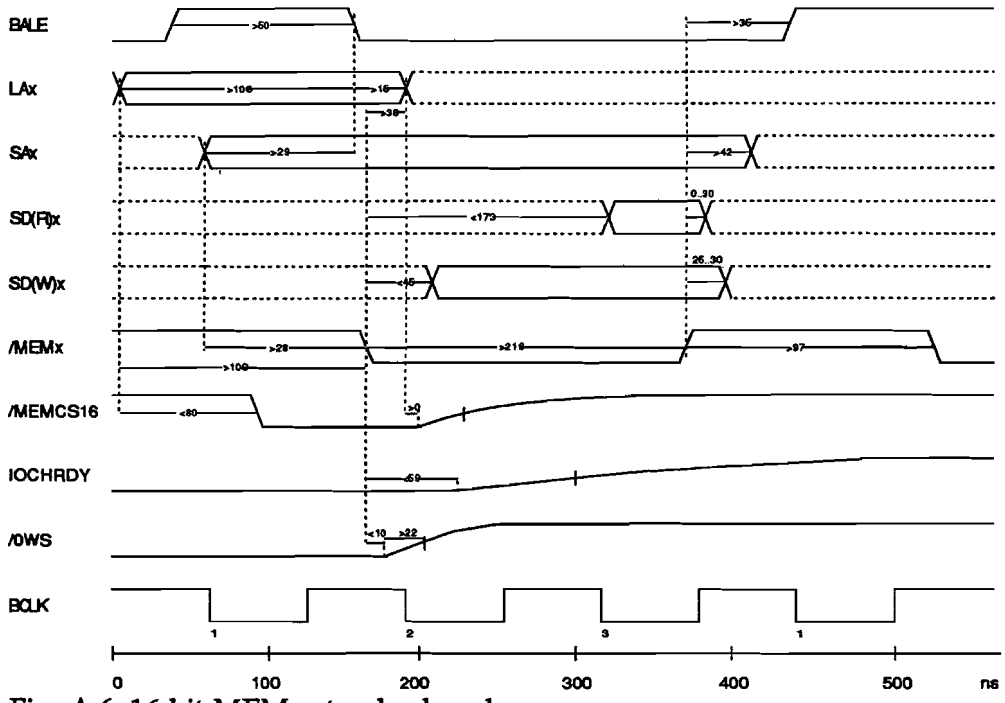


Fig. A.6: 16-bit MEMx standard cycle

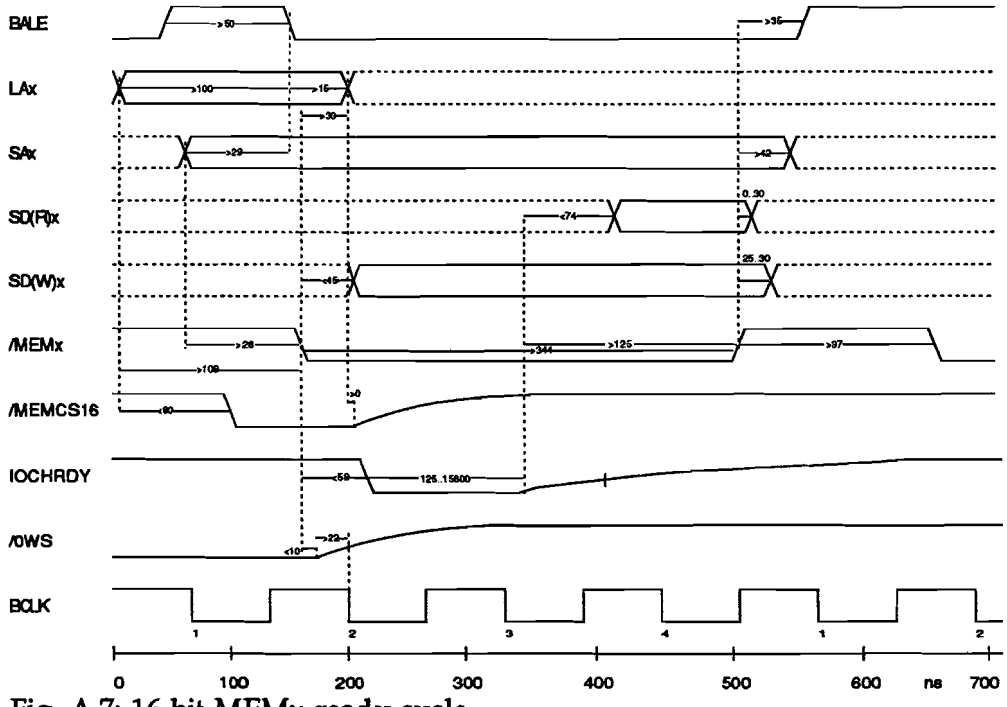


Fig. A.7: 16-bit MEMx ready cycle

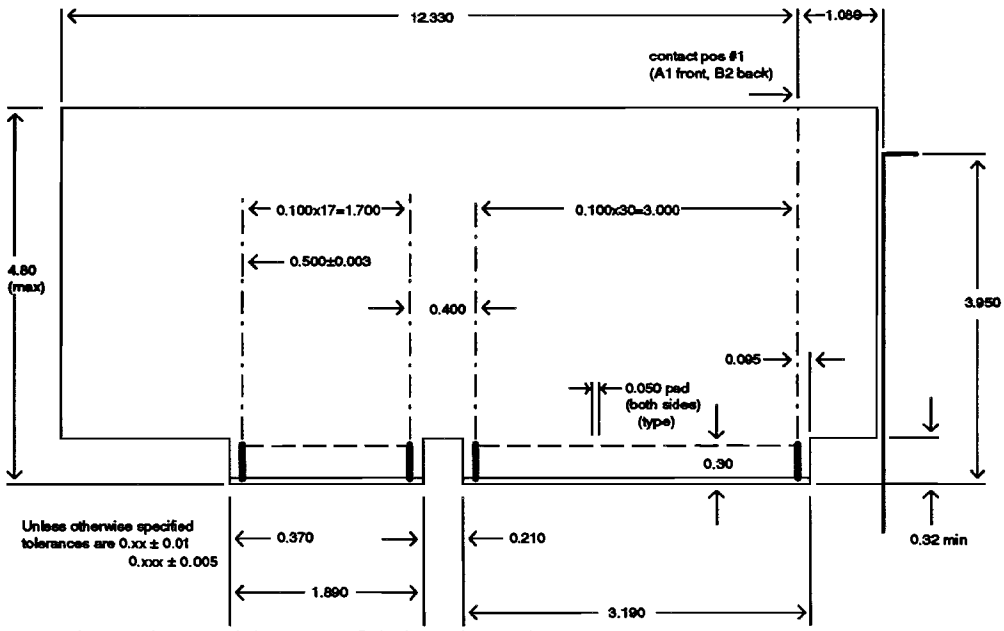
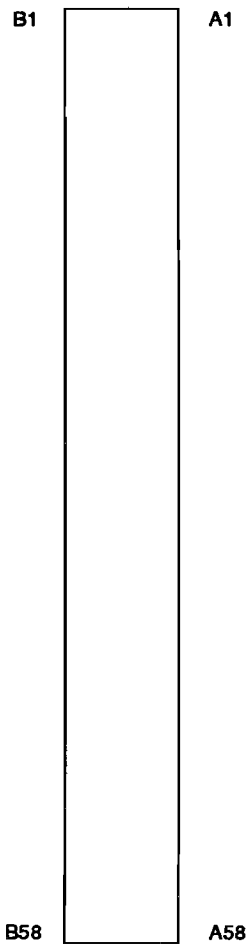


Fig. A.8: Physical layout ISA-bus board

Appendix B. VL-bus data



A: component side
B: solder side

Signal	Pin	Pin	Signal
D0	b1	a1	D1
D2	b2	a2	D3
D4	b3	a3	GND
D6	b4	a4	D5
D8	b5	a5	D7
GND	b6	a6	D9
D10	b7	a7	D11
D12	b8	a8	D13
+5 V	b9	a9	D15
D14	b10	a10	GND
D16	b11	a11	D17
D18	b12	a12	+5 V
D20	b13	a13	D19
GND	b14	a14	D21
D22	b15	a15	D23
D24	b16	a16	D25
D26	b17	a17	GND
D28	b18	a18	D27
D30	b19	a19	D29
+5 V	b20	a20	D31
A31	b21	a21	A30
GND	b22	a22	A28
A29	b23	a23	A26
A27	b24	a24	GND
A25	b25	a25	A24
A23	b26	a26	A22
A21	b27	a27	+5 V
A19	b28	a28	A20
GND	b29	a29	A18
A17	b30	a30	A16
A15	b31	a31	A14
+5 V	b32	a32	A12
A13	b33	a33	A10
A11	b34	a34	A8
A9	b35	a35	GND
A7	b36	a36	A6
A5	b37	a37	A4
GND	b38	a38	NC
A3	b39	a39	BE0#
A2	b40	a40	+5 v
NC	b41	a41	BE1#
RESET#	b42	a42	BE2#
D/C#	b43	a43	GND
M/IO#	b44	a44	BE3#
W/R#	b45	a45	ADS#
KEY	b46	a46	KEY
KEY	b47	a47	KEY
RDYRTN	b48	a48	LREADY
GND	b49	a49	LDEV<X>#
IRQ9	b50	a50	LREQ<X>#
BRDY#	b51	a51	GND
BLAST#	b52	a52	LGNT<X>#
IDO	b53	a53	+5 V
DI1	b54	a54	ID2
GND	b55	a55	ID3
LCLK	b56	a56	ID4
+5 V	b57	a57	LKEN#
LBS16#	b58	a58	LEADS#

Fig. B.1: Pin identification and signals of VL-bus

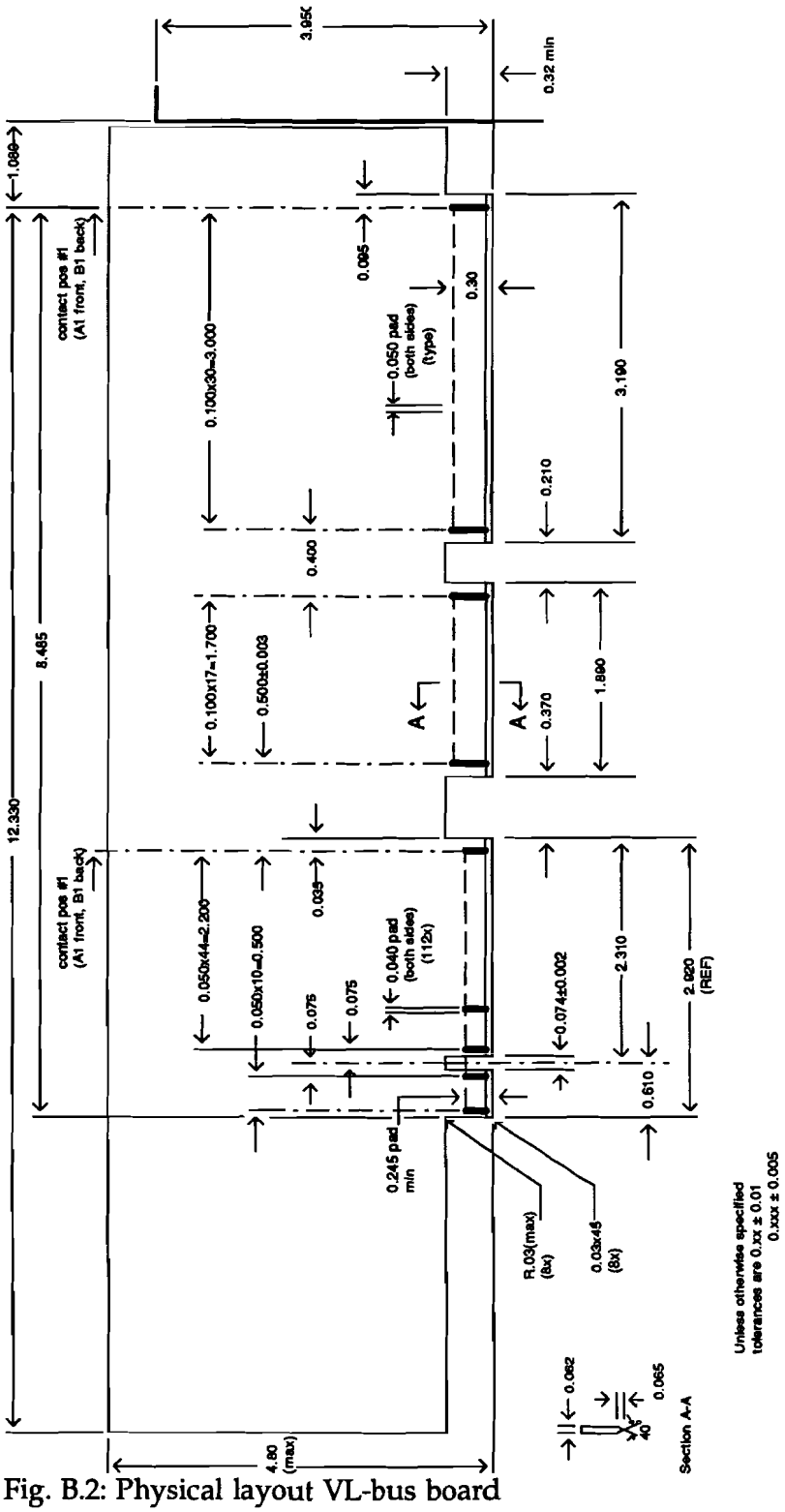


Fig. B.2: Physical layout VL-bus board

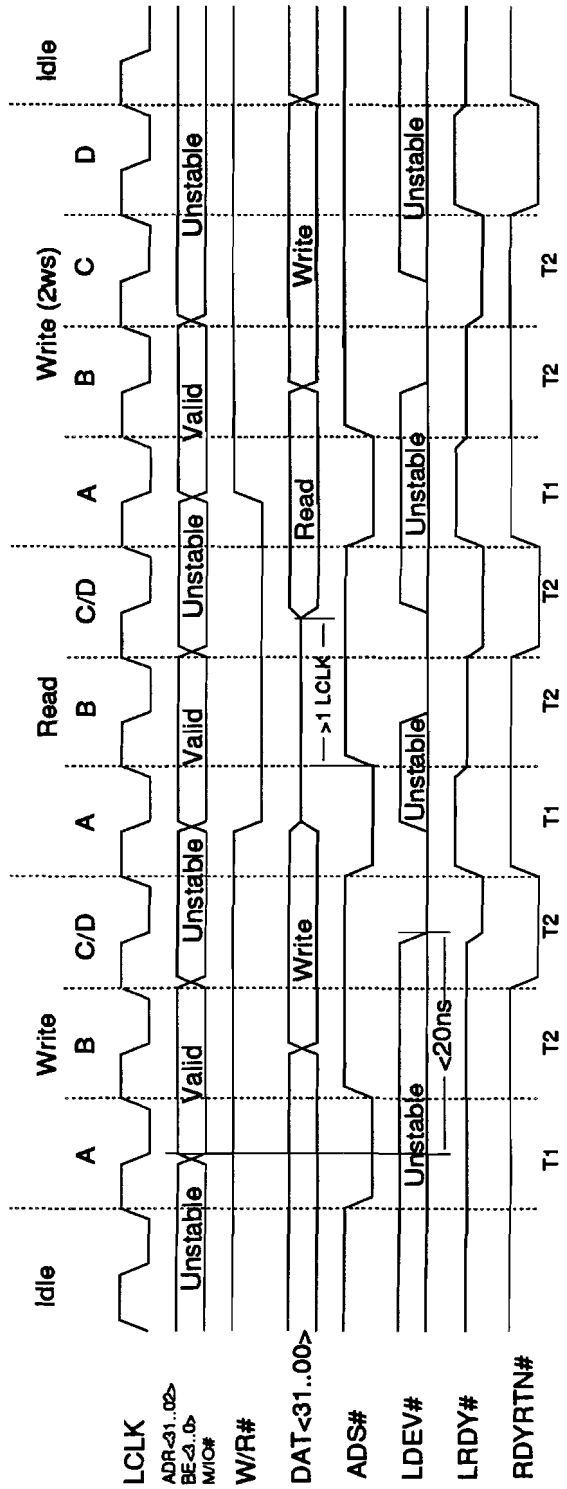


Fig. B.3: VL-bus read/write timing

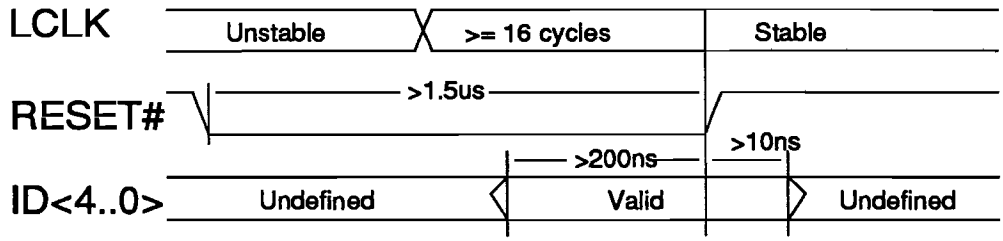
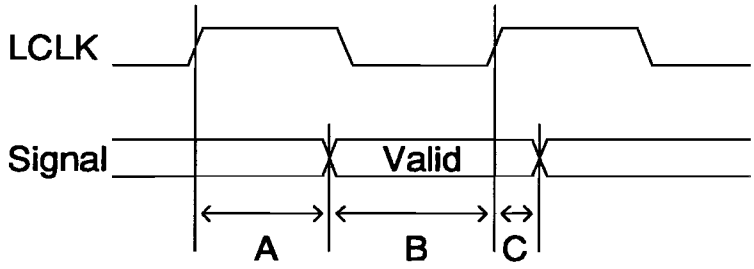


Fig. B.4: VL-bus reset timing



Signal name	A min	A max	B min	B max	C min	C max
ADR<31..02>, BE<3..0>#, M/IO#, W/R#, D/C#, ADS#, BLAST#, RESET#, RDYRTN#, LGNT<x>#, LEADS#, LKEN#	3		7	-	3	-
LRDY#, LREQ#, BRDY#, LBS16#	3	10	-	-	3	-
DAT<31..00>	3	15	7	-	3	-

Fig. B.5: Timing relative to LCLK

Table B.1 Output driver sink current requirements

	unbuffered (mA)	buffered (mA)
Controller outputs:		
LCLK	8	8
Address and data	4	8
BE<3..0>, M/IO#, W/R#, ADS#, RDYRTN#, D/C#, LEADS#, BLAST#	5	8
LGNT#	4	4
ID<4..0>	8	8
LBT outputs:		
Data bus		8
LDEV#		4
IRQ9		8
LRDY#, BRDY#, LKEN#, LBS16#		8

Appendix C. Design data

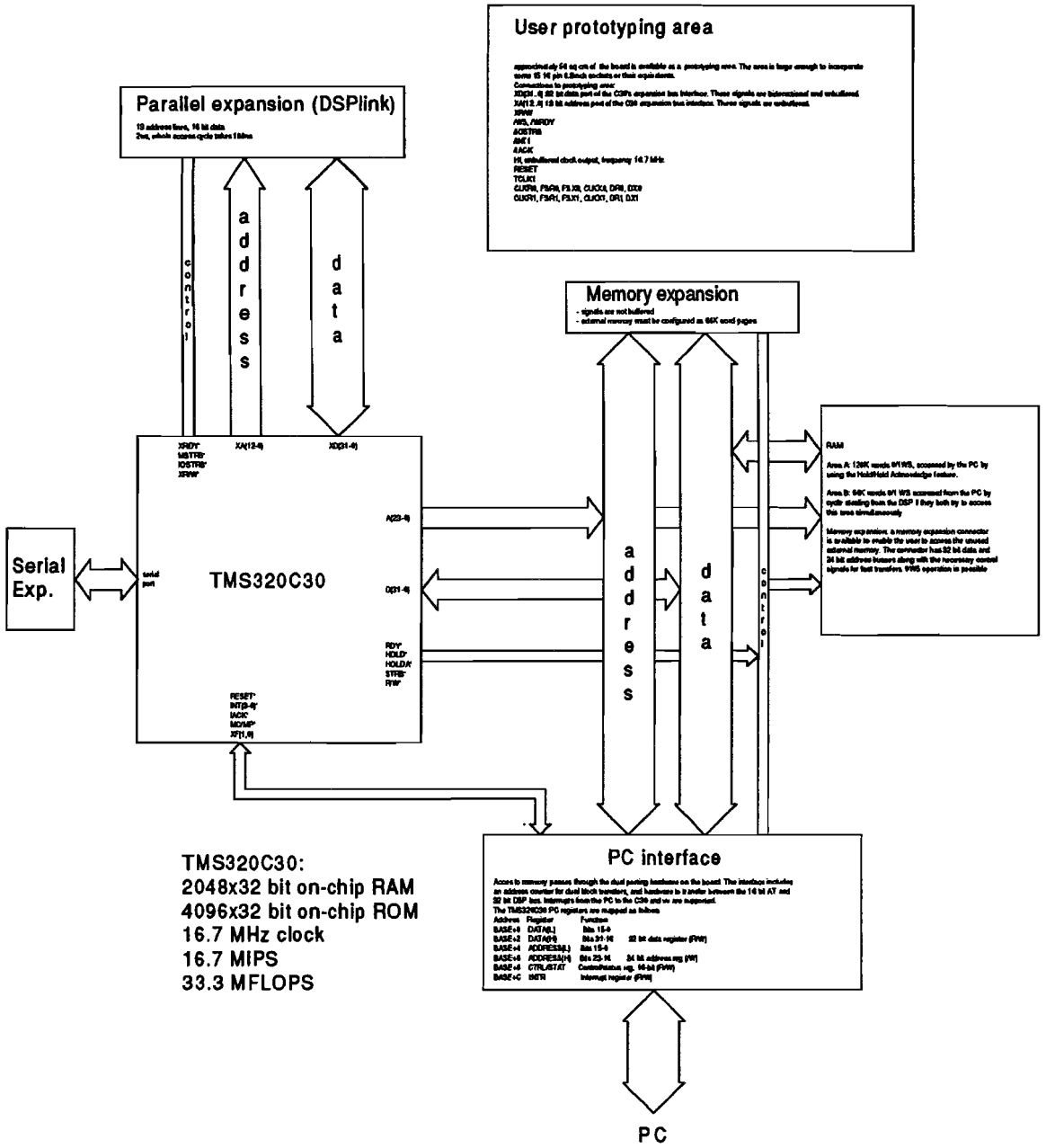


Fig. C.1: Overview TMS320C30 digital signal processor board

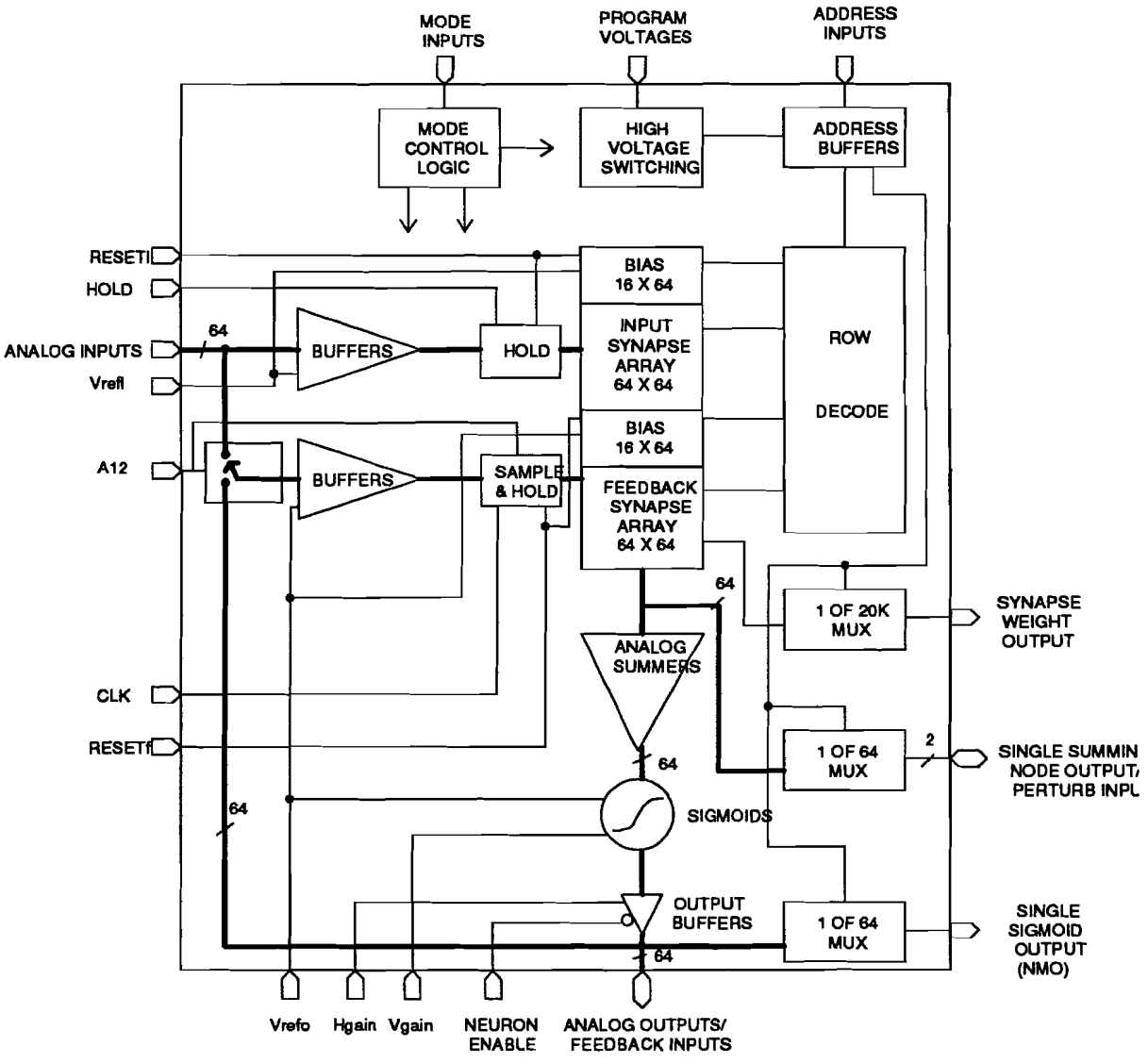


Fig. C.2: Overview Intel's ETANN chip

Characteristics of INTEL 80170NX

Symbol	Parameter	Min	Max	Units
V_{IA}	Analog input voltage	0	3.5	V
V_{OA}	Analog output voltage	0	4.0	V
V_{P1}	V_{PP1} High-voltage switch voltage	18	19	V
V_{P2}	V_{PP2} Weight modify pulse voltage	12.5	18	V
V_{REFi}	Input reference voltage	0	1.7	V
V_{REFo}	Output reference voltage	0.5	2.0	V
V_{GAIN}	Gain control voltage	0.0	5.0	V
T_{PV}	Processing delay V_{GAIN}		3	µs
T_{PH}	Processign delay H_{GAIN}		1.5	µs

IC data

Digital to analog converters (12bit)

converter	settl. time to 1/2 LSB (µs)	output	latched inputs	output range (buffered)	#circuits/IC	price (fl)	remarks
AD565AJD (AD)	0.25	current	no	fixed	1	93	
AD668JQ (AD)	0.05	current	no	0 to V_{ref}	1	160	
AD7545AKN (AD)	1	current	yes	0 to -V_{ref}	1	35	
AD7542KN (AD)	2	current	yes	0 to -V_{ref}	1	30	
AD7568BP (AD)	0.5	current	yes	0 to -V_{ref}	8	150	serial input
DAC8412EP (AD)	6	voltage	yes	V_{refl} to V_{refh}	4	134	t_{write}≥ 80ns
AD664KNuni (AD)	10	voltage	yes	0 to V_{ref}	4	172	t_{write}≥ 80ns
AD75069 (AD)	10	voltage	yes	fixed	8		t_{write}≥ 80ns

Analog to digital converters (12bit)

converter	settl. time (µs)	price (fl)
AD872JD (AD)	0.1	632
AD1671JQ (AD)	0.8	210
ADS7800JP (BB)	2.7	120
AD7572JN5 (AD)	5	91

Sample/hold devices

device	acquisition time to 0.01% (µs)	#circuits/IC	price (fl)	remars
HTC0300A (AD)	0.170	1	640	
HA1-5330-5 (HA)	0.5	1	58	
AD684JQ (AD)	1	4	125	
LF398D (PH)	8	1	4	C_H=1nF

Multiplexers

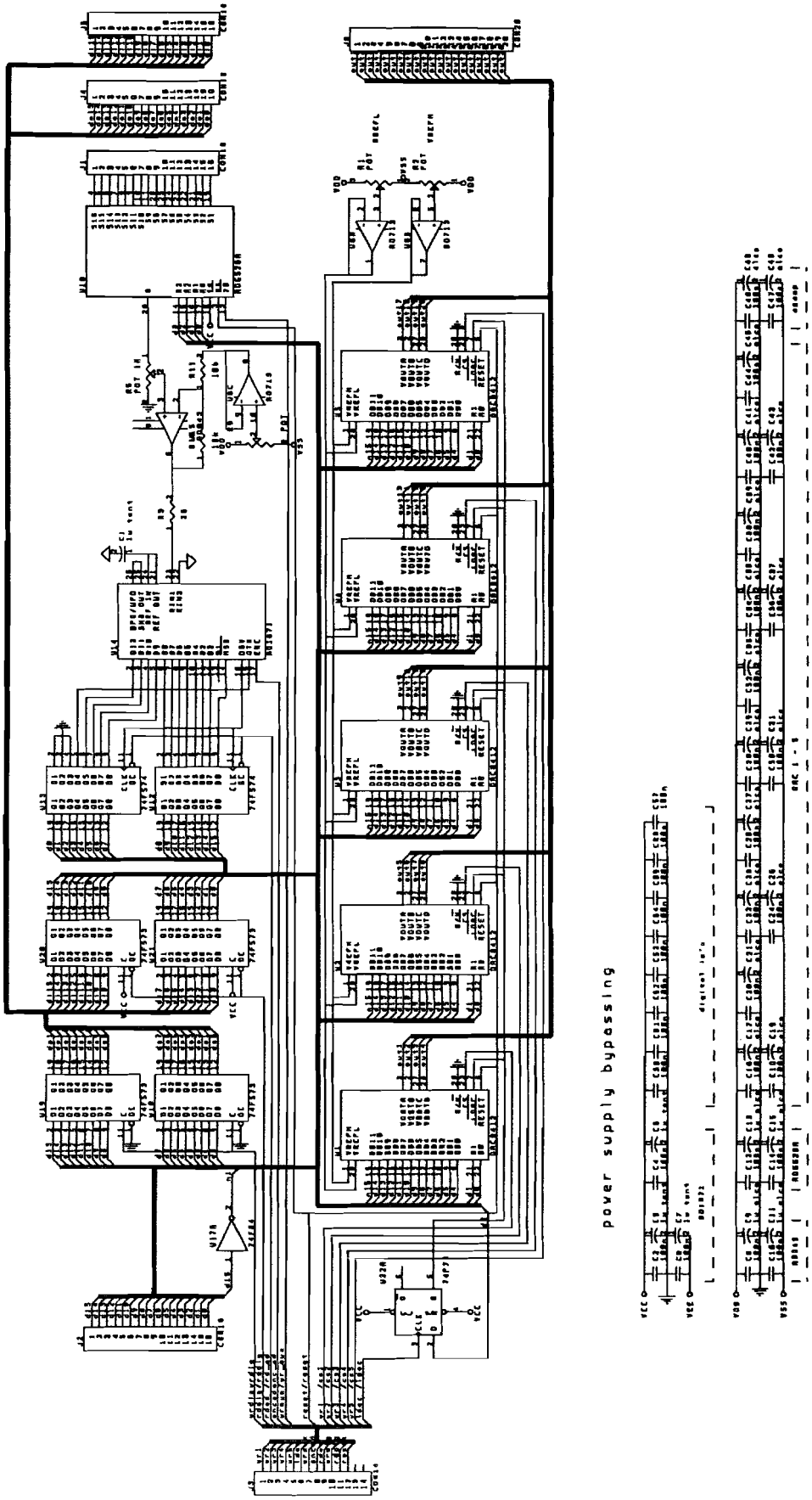
multiplexer	$t_{PD,max}$(ns)	#channels	latched inputs	price (fl)	remarks
ADG509AKN (AD)	400	2x4	no	22	
ADG529AKN (AD)	400	2x4	yes	21	t_{write}≥100ns
ADG506AKN (AD)	400	1x16	no	33	
ADG526AKN (AD)	400	1x16	yes	34	t_{write}≥100ns

Opamps

opamp	settl. time to 0.01% (µs)	V_{OS} (mV)	#circuits/IC	price (fl)
AD711JN (AD)	1	2	1	4
AD713JN (AD)	1	2	4	23
AD843JN (AD)	0.35	4	1	30

AD: Analog Devices
BB: Burr-Brown
PH: Philips

Note: given prices are a random indication and are subject to change at any moment.



C.3: Scheme analog I/O circuit

Specifications analog I/O circuit

	min	typ	max	units	remarks
POWER SUPPLIES					
V_{CC}	4.75	5.0	5.25	V	
V_{EE}	-5.25	-5.0	-4.75	V	
V_{DD}	12		15	V	15V recommended
V_{SS}	-15		-12	V	-15V recomm.
INPUTS					
input resistance		1		MΩ	
conversion time		1.3		µs	address mux valid to output valid
converter specifications:					
integral nonlinearity		±1.5	±2	LSB	
differential nonlinearity	11			bits	
offset			±8	LSB	
gain error		0.1	0.25	%FSR	
OUTPUTS					
settling time to 0.01%		6		µs	
output current I_{out}	-5		5	mA	
converter specifications:					
integral nonlinearity		0.25	±0.5	LSB	
differential nonlinearity	-1			LSB	
linearity matching dacs in IC		±1		LSB	
INPUTS/OUTPUTS					
positive reference V_H	-10.0		10.0	V	
negative reference V_L	-7.5		7.5	V	
reference range V_H-V_L	2.5			V	$V_H \geq V_L$
digital code					two's complement

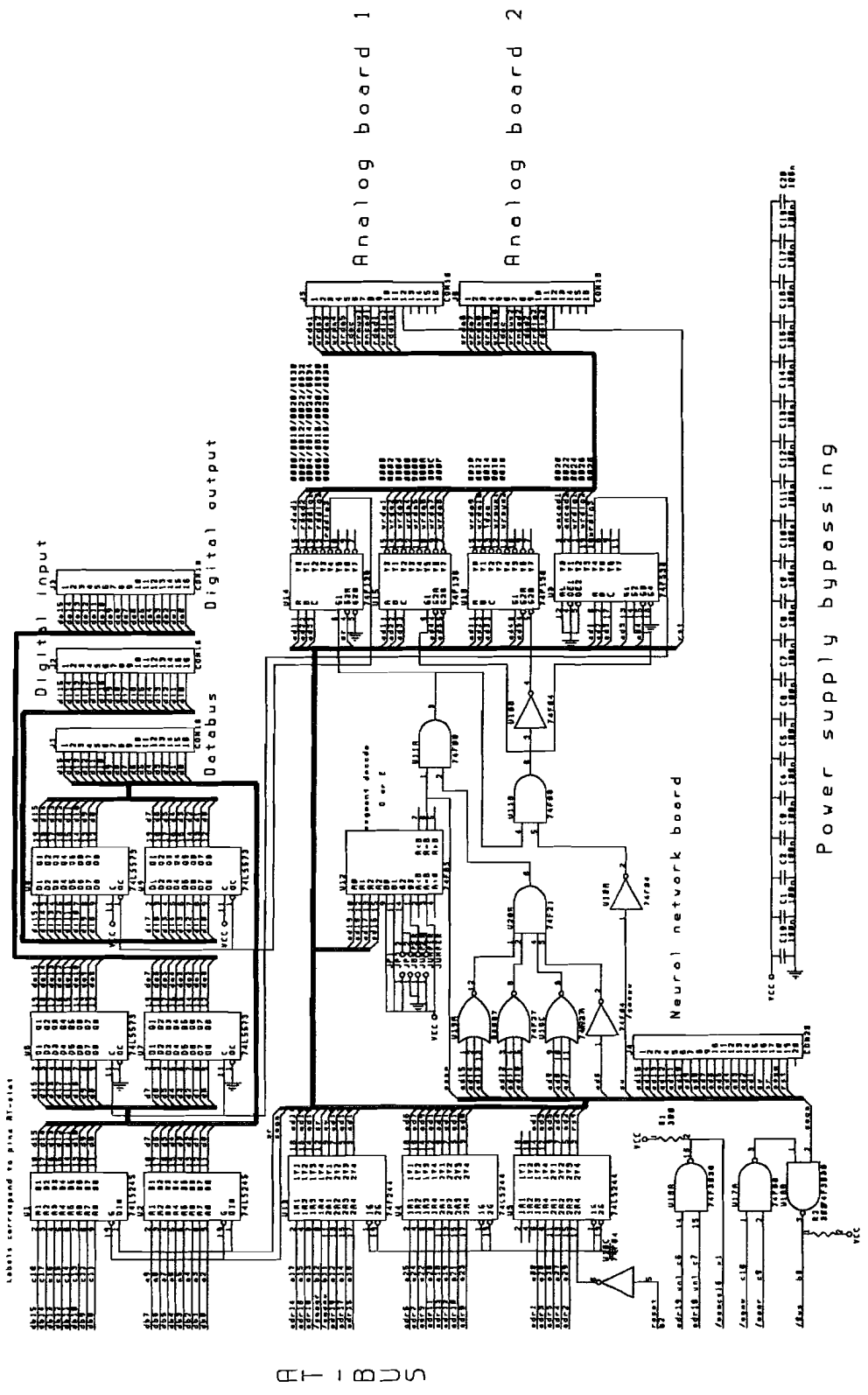


Fig. C.4: AT-bus interface circuit

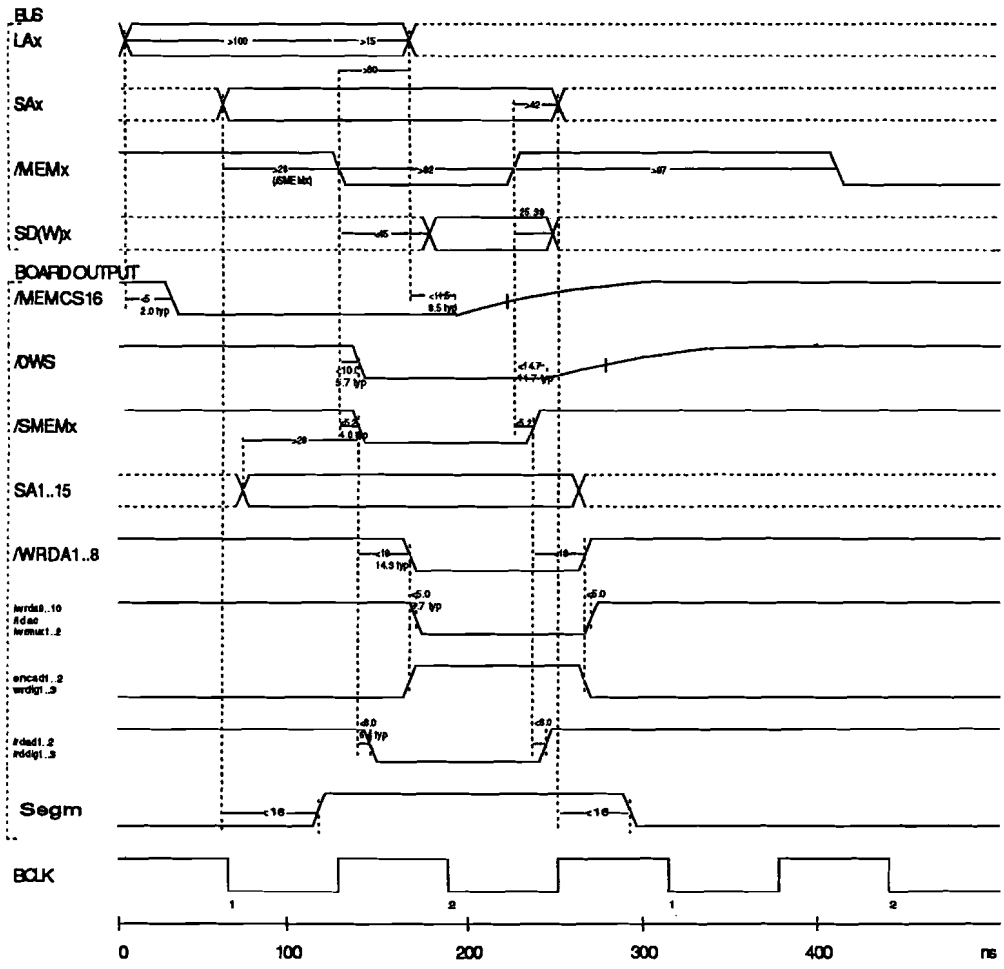


Fig. C.5: Timing AT-bus interface circuit

Physical addresses neural interface (AT-bus)

All given addresses are offsets to the chosen segment D or E (physical addresses D0000 or E0000)

Analog board 1:

D/A conversion:
DAC1 (output channels 1..4): 0000
DAC2 (output channels 5..8): 0002
DAC3 (output channels 9..12): 0004
DAC4 (output channels 13..16): 0006
DAC5 (output channels 17..20): 0008
mode flip-flop: 0014

A/D conversion:
MUX1: 0016
ADC1 (encode): 0020
ADC1 (read output): 0000/0010/0020/0030

Digital input/output:
digital output 1 (16 bit): 0024
digital input 1 (16 bit): 0004/0014/0024/0034

digital output 3 (16 bit): 0028
digital input 3 (16 bit): 0008/0018/0028/0038

Analog board 2:

DAC6 (output channels 21..24): 000A
DAC7 (output channels 25..28): 000C
DAC8 (output channels 29..32): 000E
DAC9 (output channels 33..36): 0010
DAC10 (output channels 37..40): 0012
mode flip-flop: 0014

MUX2: 0018
ADC2 (encode): 0022
ADC2 (read output): 0002/0012/0022/0032

digital output 2 (16 bit): 0026
digital input 2 (16 bit): 0006/0016/0026/0036

Neural network:
Addresses to be used by neural network: 0040-FFFF (note: only 16-bit accesses possible).

The correct DAC channel must be selected with the right address present in the word written into the DAC's latches (according to fig. 5.13). The first channel of a DAC is chosen with the last two bits equal to 00 and the fourth channel is chosen with the last two bits equal to 11. The same procedure must be followed when choosing an input channel. The last four bits of the word written to the multiplexer's latches contain the right channel (0000 for channel 0 and 1111 for channel 15).

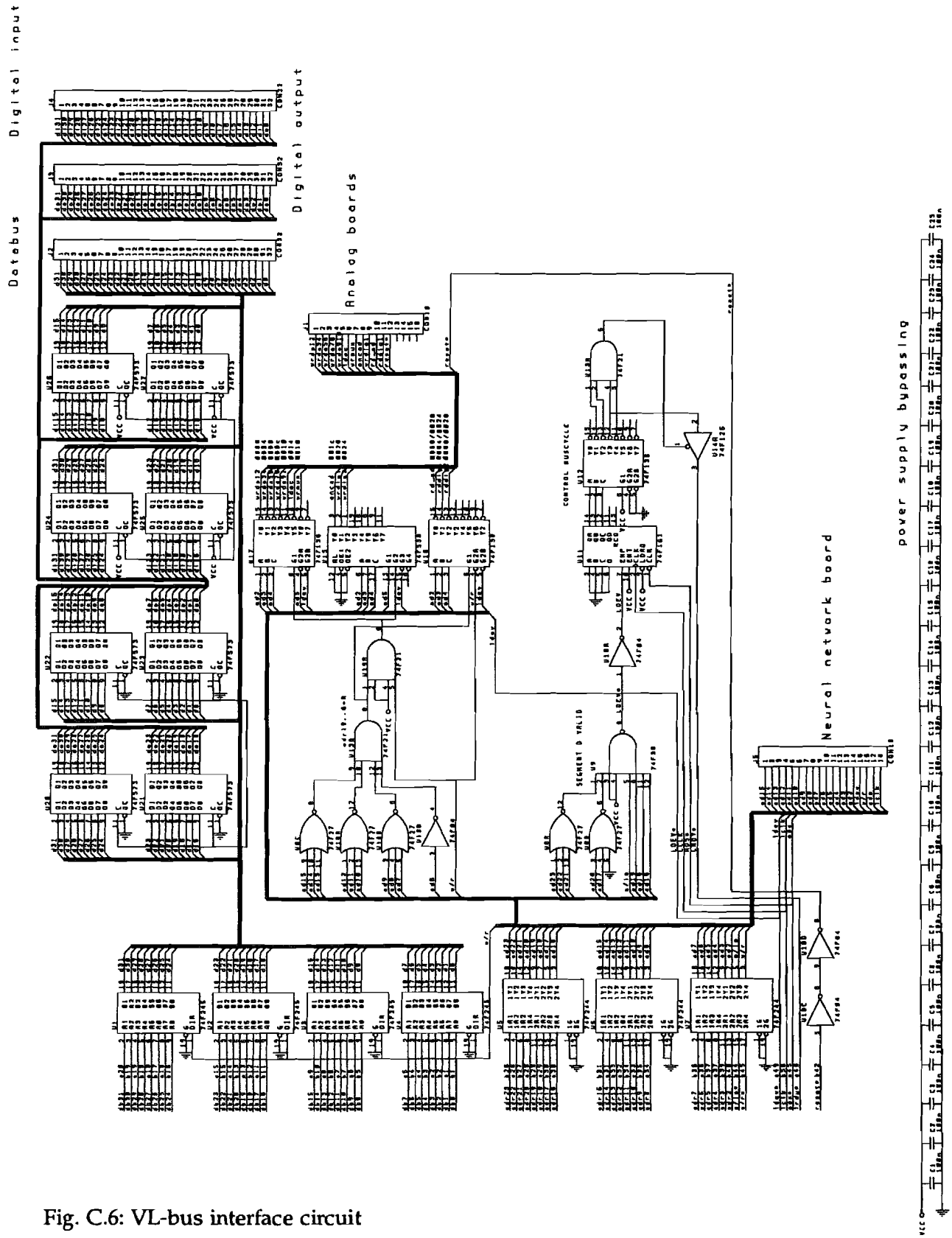


Fig. C.6: VL-bus interface circuit

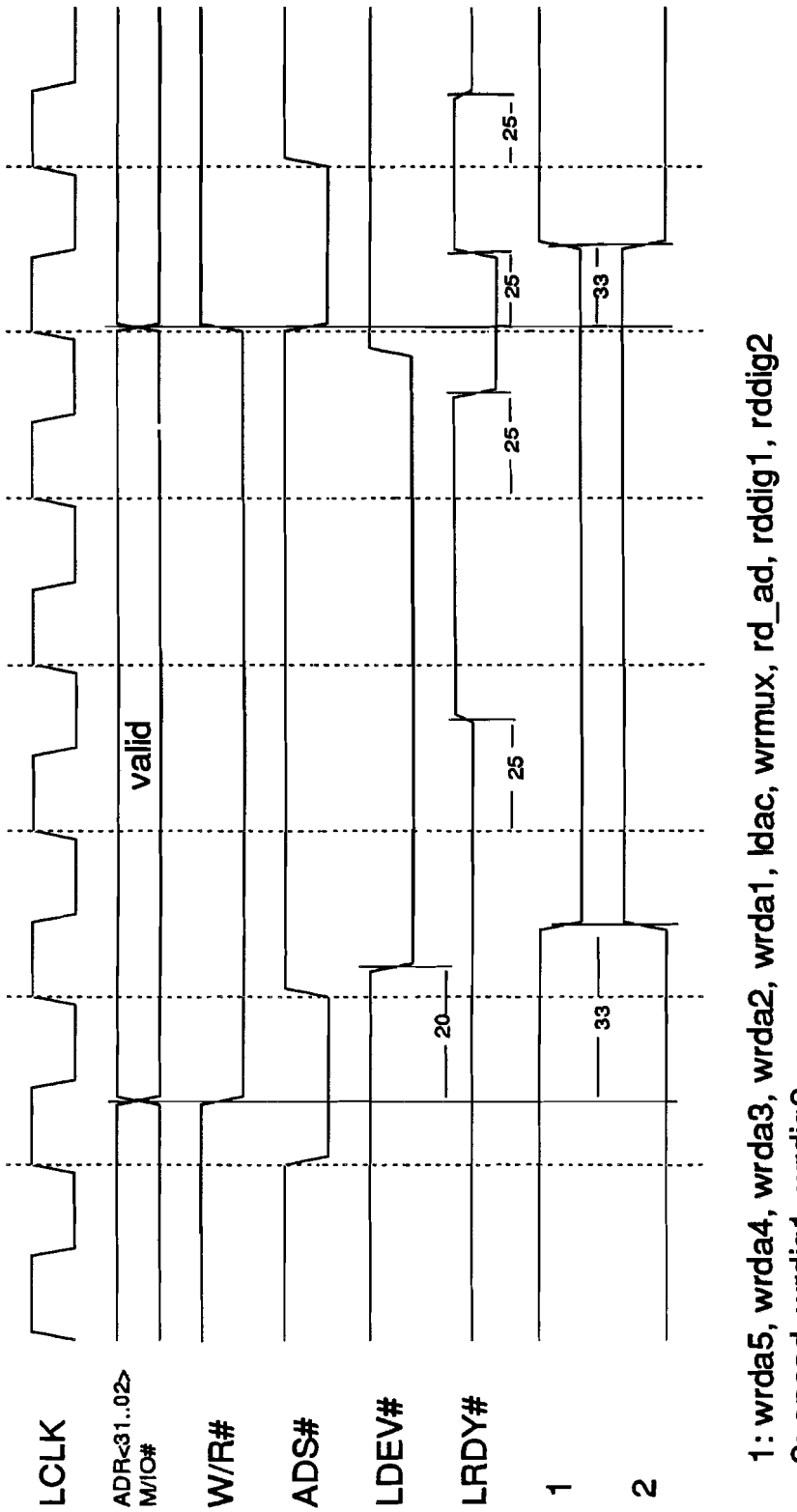


Fig. C.7: Timing VL-bus interface circuit

Physical addresses neural interface (VL-bus)

All given addresses are offsets to the chosen segment D (physical address D0000)

D/A convertersion:
DAC1/6 (output channels 1..8): 0000
DAC2/7 (output channels 9..16): 0004
DAC3/8 (output channels 17..24): 0008
DAC4/9 (output channels 25..32): 000C
DAC5/10 (output channels 33..40): 0010
mode flip-flop: 0014

A/D conversion:
MUX1/2: 0018
ADC1/2 (encode): 001C
ADC1/2 (read output): 0000/0020

Digital input/output:
digital output 1 (32 bit): 0020
digital input 1 (32 bit): 0004/0024

digital output 2 (32 bit): 0024
digital input 2 (32 bit): 0008/0028

Neural network:
Addresses to be used by neural network: 0040-FFFF (note: only 32-bit accesses possible).

The correct DAC channels must be selected with the right address present in the word written into the DAC's latches (according to fig. 5.13). The first channel of a DAC is chosen with the last two bits equal to 00 and the fourth channel is chosen with the last two bits equal to 11. The same procedure must be followed when choosing an input channel. The last four bits of the word written to the multiplexer's latches contain the right channel (0000 for channel 0 and 1111 for channel 15).

Appendix D. Software

Conversion functions

```
float integer_to_float(int integer, float lower, float upper)
{ /* convert integer containing 12-bit two's complement code to actual voltage */
        float   zero, step;

        zero=(upper-lower)/2+lower; /* zero voltage */
        step=(upper-lower)/65536; /* step size in case of 16-bit code */
        return (integer*step+zero);
}

int float_to_integer(float floating, float lower, float upper)
{ /* convert floating point value between lower and upper to integer according to */
  /* format required by analog I/O board */
        float   zero, step;
        int     i;

        zero=(upper-lower)/2+lower; /* zero voltage */
        step=(upper-lower)/65536; /* step in case of 12-bit code */
        i= (floating-zero)/step; /* conversion to 16-bit integer */
        return (i/16)*16; /* return in special format required by analog I/O board (quantized to 12-bit) */
}
```

Analog output functions

```
void load_single_dac (int channel, int value)
{       /* load channel (0-31) with value (format according to fig. 5.13) */
        int     far *daptr;

        daptr=dac_base+2*(channel/4)   /* set pointer to right IC */
        *daptr=value+(channel%4);   /* send value to DAC (number of DAC in IC is added with channel%4) */
}

void load_all_dacs(int *value, int number)
{       /* value must point to first element of array of a total of number integer values */
        int     far *daptr;
        int     ic=0, dachannel=0, i; /* ic indicates DAC IC, dachannel indicates one of four DACs in that IC */

        daptr=dac_base; /* setpointer to first DAC IC */
        for (i=0; i<number; i++)
        {       *daptr=*(value+i)+dachannel;
                /* one of four channels (indic. by dachannel) of converter IC (indic. by ic) is loaded with value*/
                dachannel=(dachannel+1)%4;
                if (dachannel%4==0)
                {       /* all dacs in single DAC IC loaded */
                        ic+=2;
                        daptr+=ic; /* set pointer to next DAC IC */
}       }       }
```

```
void update_dacs(void)
{       /* update outputs of all DAC ICs */
        int     far *daptr;

        daptr=dac_base;
        *daptr=1; /* set update mode */
        *daptr=0; /* set load mode again */
}
```

Analog input functions

```
int read_single_adc (int channel)
{       /* read single input channel (0..31) */
        int     far *muxptr, far *adptr, i;

        muxptr=2*(channel/16)+mux_base; /* set pointer to address right multiplexer IC */
        *muxptr=channel%16; /* choose right mux channel (one of sixteen) */
        adptr=2*(channel/16)+adc_base; /* set pointer to address right ADC IC */
        *adptr=0; /* write dummy word to start conversion */
        for (i=0;i<20;i++); /* delay of about 600 ns, the right number in the for loop must be determined in practice */
        return *adptr; /* read result of conversion */
}

void read_all_adcs(int *value, int number)
{       /* value must point to array of total of number integers, number must be even */
        int     far *muxptr1, far *muxptr2, far *adptr1, far*adptr2;
        int     i, j, channel=0;

        muxptr1=mux_base;           muxptr2=mux_base+2; /* set pointers to addresses multiplexer ICs */
        adptr1=adc_base;            adptr2=adc_base+2; /* set pointers to addresses ADC ICs */
        *muxptr1=*muxptr2=channel;  /* set mux channels */
        *adptr1=*adptr2=0; /* start conversions */
        for (i=0; i<number/2; i++)
        {
                channel+=1;
                *muxptr1=*muxptr2=channel; /*set new channel of multiplexers*/
                *(value+channel-1)=*adptr1; /* read results of conversions, no delay has to be inserted because two */
                *(value+channel+15)=*adptr2; /* two buscycles are needed to set new channels of multiplexers */
                *adptr1=*adptr2=0; /* start new conversion */
        }
}
```

```
void process_all_dacs_adcs(int *dacvalues, int *adcvalues)
{   /* dacvalues is pointer to array of 32 integers to be output, adcvalue is pointer to array of 32 integers to be read */
        int     far *daptr, far *muxptr1, far *muxptr2, far *adptr1, far *adptr2;
        int     ic=0, adchannel=0, dachannel=0, i, j;

        muxptr1=mux_base;           muxptr2=mux_base+2; /* set pointers to addresses multiplexer ICs */
        adptr1=adc_base;            adptr2=adc_base+2; /* set pointers to addresses ADC ICs */
        daptr=dac_base; /* set pointer to address first DAC IC */
        *muxptr1=*muxptr2=adchannel; /*set mux channel */
        *adptr1=*adptr2=0; /* start conversion */
        for (i=0, i<16; i++)
        {
                adchannel+=1;
                *muxptr1=*muxptr2=adchannel;  /* set new mux channel */
                *daptr=*(dacvalues+2*i)+dachannel; /* load latch of DAC, right channel in single DAC IC is added */
                *daptr=*(dacvalues+2*i+1)+dachannel+1; /* with dachannel */
                dachannel+=2;
                if (dachannel%4==0)
                { /* all four latches in single DAC IC loaded */
                        ic+=2;
                        daptr+=ic; /* goto next dac IC */
                }
                *(adcvalues+adchannel-1)=*adptr1; /* read result of a/d conversion and place the result */
                *(adcvalues+adchannel+15)=*adptr2; /* in adcvalue */
        }
        daptr=dac_mode; /* set pointer to address mode flip-flop */
        *daptr=1; /* set update mode */
        *daptr=0; /* return to load mode again */
}
```

Digital input and output functions

```
void write_digital(int number, int value)
{ /* write value into one of three digital latches */
  /* number=0,1,2 */
        int     far *digptr;

        digptr=dig_base+2*number; /* set pointer to address right latch */
        *digptr=value; /* write latch */
}

int read_digital(int number)
{ /* read one of three latches */
        int     far *digptr;

        digptr=dig_base+2*number; /* set pointer ro address right latch */
        return *digptr; /* read latch */
}
```